DISCRETE SIMULATIE

met een inleiding in SIMULA

P.A.M. Griep en
S.D.P. Flapper



ACADEMIC SERVICE

Discrete Simulatie
met een inleiding in SIMULA

DISCRETE SIMULATIE

met een inleiding in SIMULA

P.A.M. Griep en
S.D.P. Flapper



ACADEMIC SERVICE

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Griep, P.A.M.

Discrete simulatie met een inleiding in SIMULA / P.A.M. Griep,
S.D.P. Flapper. - Schoonhoven : Academic Service
ISBN 90-6233-272-2
SISO 521 SVS 8.12.3 UDC 681.3.001.572:800.92 Simula
NUGI 852
Trefw.: SIMULA (programmeertaal) / computersimulatie.

Uitgegeven door: Academic Service
Postbus 81
2870 AB Schoonhoven

Zetwerk: tedejo producties, Voorhout
Dit boek is zetklaar gemaakt met de programma-editor van WordPerfect Library, gezet met LaTeX (PCTeX) en afgedrukt op een Cordata LP300X laserprinter
Omslagontwerp: Olivier
Druk: Krips Repro Meppel
Bindwerk: Meeuwis, Amsterdam

ISBN 90-6233-272-2
NUGI 852

Voorwoord

Dit boek behandelt 'discrete simulatie' zoals dat gegeven wordt binnen de collegecyclus 'discrete simulatie en wachttijdproblemen' aan zowel tweede jaars informatica - als derde jaars bedrijfskundestudenten van de Technische Universiteit Eindhoven.

Door de elementaire behandeling van de verschillende onderdelen heeft men geen specifieke voorkennis nodig, afgezien van enige basiskennis op de gebieden programmeren en statistiek.

Hierdoor lijkt dit inleidende boek tevens geschikt voor het HBO-onderwijs en voor zelfstudie.

Aan de samenstelling van dit boek alsmede de hieraan ten grondslag liggende collegedictaten is door verscheidene personen een bijdrage geleverd. Hiervoor allen hartelijk dank.

Van de studentassistenten met name ing. J. Geelen en R. Achterberg; van de collega's met name ir. H. J. Pels (SIMULA), ing. R. Vodegel, ing. Th. Sanders, prof.dr. P. C. Sander (statistiek), dr. ir. J. Geurts (wachttijdtheorie) en van de HEAO te Arnhem dr. S. A. de Wit. Voor de tekstverwerking zorgden de secretaresses P. v. d. Donk, J. Loonen, Y. v. d. Veer en C. van Woensel.

Eindhoven, mei 1987

P.A.M. Griep
S.D.P. Flapper

Inhoud

Hoofdstuk 1

Inleiding

1.1 Waarom simuleren

Iedereen heeft in zijn/haar leven te maken met (het oplossen van) problemen. Problemen treden in het algemeen op als gevolg van (impliciet) gestelde doelen, welke tevens de gewenste oplossing van het probleem aangeven. Sommige problemen kunnen 'direct' opgelost worden; de oplossingen zijn bekend. Bijvoorbeeld het probleem van een 'lekke fietsband' (zie linker tak figuur 1.1).

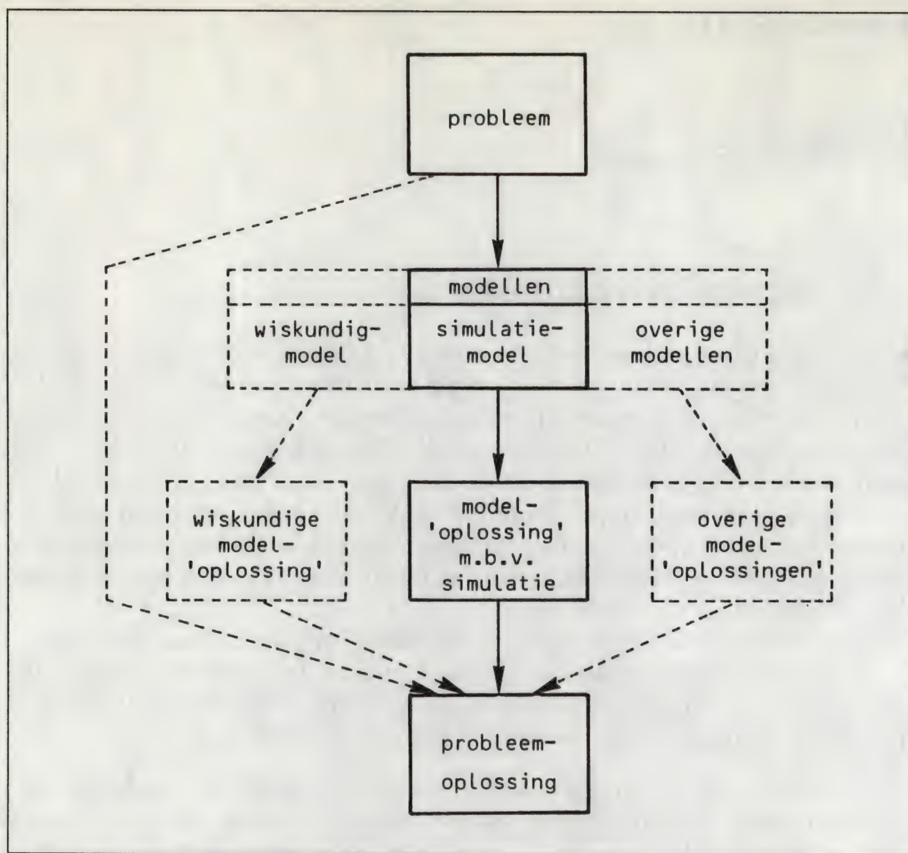
Andere problemen lijken of zijn nieuw. Bij het zoeken naar een oplossing hiervan kan en moet men soms gebruik maken van modellen (dat wil zeggen vereenvoudigde voorstellingen van de werkelijkheid) van de systemen waarin de problemen zich voordoen.

Door het gebruik van modellen wordt het mogelijk een stuk werkelijkheid en eventuele veranderingen van die werkelijkheid te bestuderen zonder dat een (ongewenste) ingreep in die werkelijkheid nodig is. Bijvoorbeeld onderzoek naar de optimale opstelling van machines in een produktiehal.

Indien het gedrag van een systeem met betrekking tot een probleem goed beschreven kan worden met behulp van wiskundige vergelijkingen die men exact of benaderd kan oplossen dan zal men veelal wiskundige modellen toepassen. Denk bijvoorbeeld aan de bewegingen van macroscopische voorwerpen met behulp van de wetten van de mechanica, de econometrische modellen van het centraal planbureau of de diverse wiskundige modellen uit de operations research. Voor een aantal probleemsituaties kan dit (nog) niet of zou dit te veel tijd en/of geld kosten. Bijvoorbeeld voor het bepalen van geschikte prioriteitsregels voor het in bewerking nemen van orders in een zogenaamde job-shop (Baker & Bertrand 1981).

In dergelijke situaties kan men van de te beschouwen systemen vaak wel een simulatiemodel maken.

Onder een **simulatiemodel** *verstaan we een zodanige beschrijving van de componenten van een systeem en hun onderlinge relaties dat op grond hiervan het gedrag van het systeem (in de tijd) nagebootst kan worden.*



Figuur 1.1: Het oplossen van problemen

Voorwaarden voor het kunnen maken van zo'n (simulatie)model is dat er een concrete doelstelling geformuleerd is en dat de voor het probleem essentiële eigenschappen van de systeemcomponenten kwantificeerbaar zijn. Voor het begrip simulatie zullen we de volgende definitie hanteren (gebaseerd op Shannon 1975):

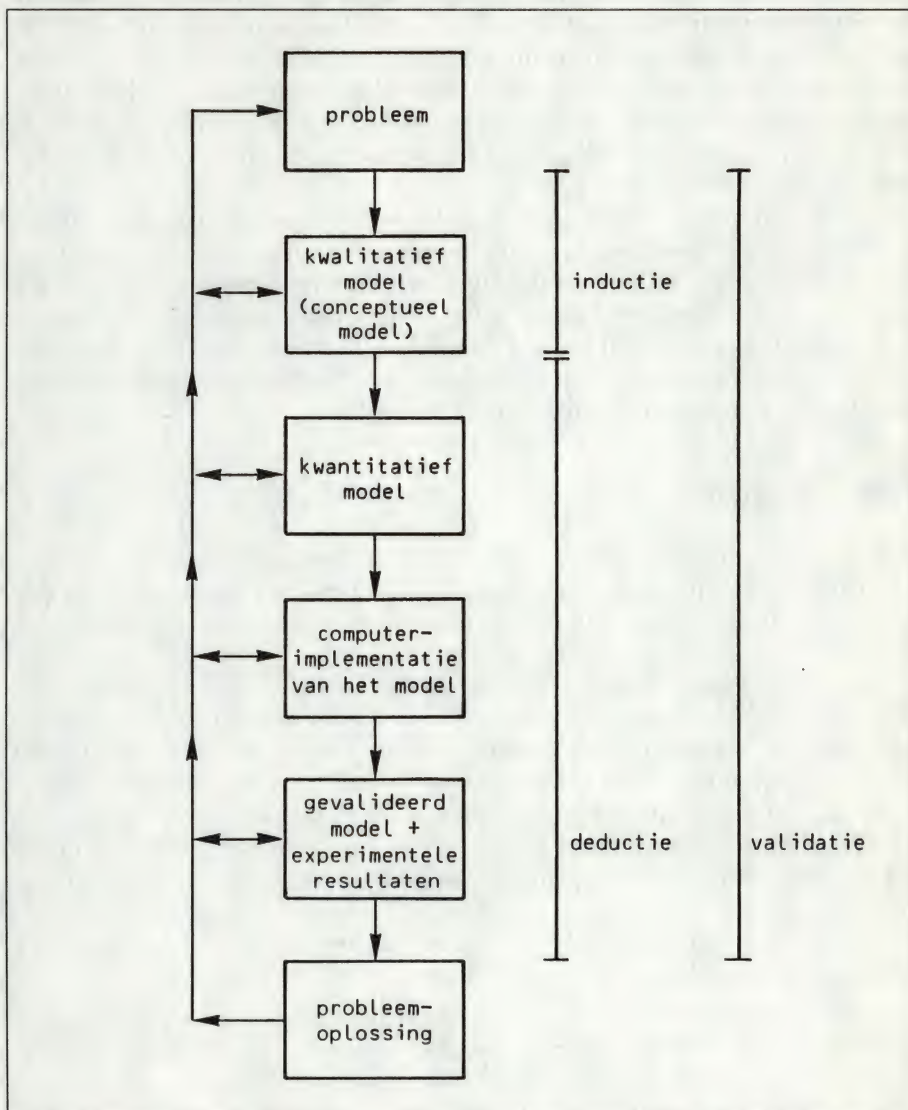
> **Simulatie** *is het proces om van een reëel systeem een simulatiemodel te ontwikkelen en met dat model experimenten uit te voeren om zodoende inzicht te krijgen in het (toekomstig) gedrag van dat systeem onder verschillende omstandigheden.*

Voor de volledigheid merken we op dat voor het oplossen van sommige problemen ook andersoortige modellen geschikt kunnen zijn; zie rechter tak figuur 1.1 (bijvoorbeeld het model van Bohr waarin de onderdelen van een atoom als harde bolletjes worden voorgesteld).

Voorbeelden van toepassingen van simulatiemodellen zijn:

- (schaal)model van Zeeland om inzicht te krijgen in het stromingspatroon binnen de zeearmen,
- CAD (computer aided design) modellen, bijvoorbeeld bij het ontwerpen van auto's, kleding, de layout van een machinefabriek etc.,

Bij meer structureel gebruik van simulatiemodellen spreekt men wel van Beslissings Ondersteunende Systemen (BOS) of Decision Support Systems (DSS). De hoofdcomponenten van dit soort systemen zijn: een modellenbank, een gegevensbank van de (input)gegevens voor de modellen en een mens-machine interface voor een flexibel gebruik van het systeem.



Figuur 1.2: Het oplossen van een probleem met behulp van computersimulatie

Wij zullen ons beperken tot simulaties op een computer (Pidd 1984, Law & Kelton 1982, Spriet & Vansteenkiste 1982, Swartz 1984, Neelamkavil 1987).

Met betrekking tot het gedrag van simulatiemodellen in de tijd kunnen we een onderscheid maken tussen continu en discreet.

Binnen een continu simulatiemodel verandert de toestand van een systeem continu. Typerend voor de beschrijving van het gedrag van de componenten is het gebruik van differentiaalvergelijkingen. Oorspronkelijk werd dit soort simulaties uitgevoerd op analoge computers. Later werden er technieken en hulpmiddelen ontwikkeld om het gedrag op digitale computers na te bootsen. Bij discrete simulatiemodellen verandert de toestand van het systeem op discrete tijdstippen; deze modellen zijn bij uitstek geschikt voor implementatie op digitale computers. Bij het modelleren van problemen op bijvoorbeeld bedrijfskundig, economisch, sociaal en technisch gebied kan het tijdsgedrag dikwijls discreet behandeld worden. We zullen ons verder tot dit soort systemen beperken.

Binnen tal van systemen treden wachttijden (en wachtrijen) op, bijvoorbeeld bij kassa's, machines, op verkeerswegen, bij verwerking van computerjobs. Iedereen kent ze uit eigen ervaring. Daarom zullen we aan de hand van een eenvoudig wachtrijprobleem de in figuur 1.2 geschetste aanpak behandelen en toelichten. Het gaat daarbij om een tankstation met een beperkte wachtruimte waarbij men geïnteresseerd is in de wachttijden van klanten onder verschillende condities (een nadere omschrijving volgt in hoofdstuk 2).

1.2 Aanpak

Het oplossen van een probleem met behulp van computersimulatie is een cyclisch proces waarin meer dan eens teruggekoppeld kan moeten worden (zie figuur 1.2). In dit proces kunnen een aantal fasen worden onderscheiden: de inductie-, de deductie- en de validatiefase.

In de inductiefase vindt de feitelijke modelbouw plaats. In de deductiefase vooral het experimenteren met het model om tot een gewenste oplossing te komen. De 'validatiefase' verloopt parallel aan beide voornoemde fasen en betreft de activiteiten nodig om de geldigheid van het model te waarborgen.

In hoofdstuk 2 komt na een korte, algemene behandeling van systemen en modellen het opstellen van een kwalitatief-beschrijvend of conceptueel model aan de orde. Samen met de gebruiker worden hierin de specificaties van het model vastgelegd.

Bij de beschrijving van het gedrag van dit model in de tijd zijn meerdere uitgangspunten mogelijk:

a. De activiteitenbeschrijving; een beschrijving van de afzonderlijke activiteiten met de voorwaarden waaronder deze optreden alsmede de er door veroorzaakte toestandsveranderingen van het systeem.
b. De eventbeschrijving; het uitgangspunt daarbij is dat een beschrijving wordt gegeven van alle veranderingen van de toestand van het systeem

op de tijdstippen dat bepaalde gebeurtenissen (events) plaatsvinden.

c. De procesbeschrijving; hierbij worden de toestandsveranderingen van een systeem beschreven vanuit de systeemcomponenten die de veranderingen veroorzaken.

Voor laatstgenoemde beschrijvingswijze neemt de belangstelling steeds meer toe, ook bij het specificeren van 'gewone' informatiesystemen (Jackson 1983, Cameron 1986). De procesbeschrijving zal in de rest van dit boek centraal staan. Bij het specificeren van de procesbeschrijving zal een stroomschematekentechniek gebruikt worden.

Om tot een kwantitatief model te komen moeten daarnaast de waarden van de benodigde parameters uit de te simuleren werkelijke situatie worden bepaald.

Om inzicht te krijgen in het simulatieproces op implementatienivo zullen we in hoofdstuk 3 het tankstationsysteem eerst handmatig simuleren.

In hoofdstuk 4 wordt ingegaan op de statistische aspecten van simulatie waaronder de betrouwbaarheid van simulatieresultaten.

In hoofdstuk 5 zal worden nagegaan in hoeverre de verschillende kwantitatieve modellen met behulp van een programmeertaal als PASCAL kunnen worden geïmplementeerd op een computer. Het blijkt dat voor implementatie van procesbeschrijvingen talen zoals PASCAL minder geschikt zijn.

Op procesbeschrijvingen geënt zijn de zogenaamde object-georiënteerde talen. In hoofdstuk 6 zal, als voorbeeld, de reeds 'klassieke' taal SIMULA beknopt behandeld worden. Vervolgens wordt de procesbeschrijving van het tankstation in SIMULA vertaald.

In hoofdstuk 7 wordt volgens de in dit boek behandelde methode uitgaande van de procesbeschijving, een wat uitvoeriger probleem tot in detail uitgewerkt en daarna geimplementeerd in SIMULA.

In hoofdstuk 8 worden van een aantal opgaven uitwerkingen gegeven.

Tenslotte volgt in hoofdstuk 9 een overzicht van de literatuur waarnaar in dit boek verwezen wordt.

Het voorgaande vormt tevens een geschikte basis voor het gebruik van andere simulatietalen en simulatiepakketten. Binnen het kader van dit boek wordt hier verder geen aandacht aan besteed.

1.3 Opgaven

1 *Bedenk zelf enkele problemen die zich wel c.q. niet lenen om opgelost te worden met behulp van simulatie. □

* (Van de opgaven voorzien van * is in hoofdstuk 8 een uitwerking opgenomen)

Hoofdstuk 2

Systeembenadering en beschrijvingswijzen van discrete modellen

2.1 Inleiding

Voor het afbakenen en structureren van een probleem(gebied) zal gebruik gemaakt worden van de systeembenadering. Deze systeembenadering is eveneens van belang voor het modelleren van het reële systeem. Het tijdsaspect van de zo verkregen discrete modellen kan op een drietal wijzen worden beschreven: de activiteiten-, de event- en de procesbeschrijving. Zo komen we tot het kwalitatieve of conceptueel model, dat de basis vormt voor het kwantitatieve model.

2.2 Systemen

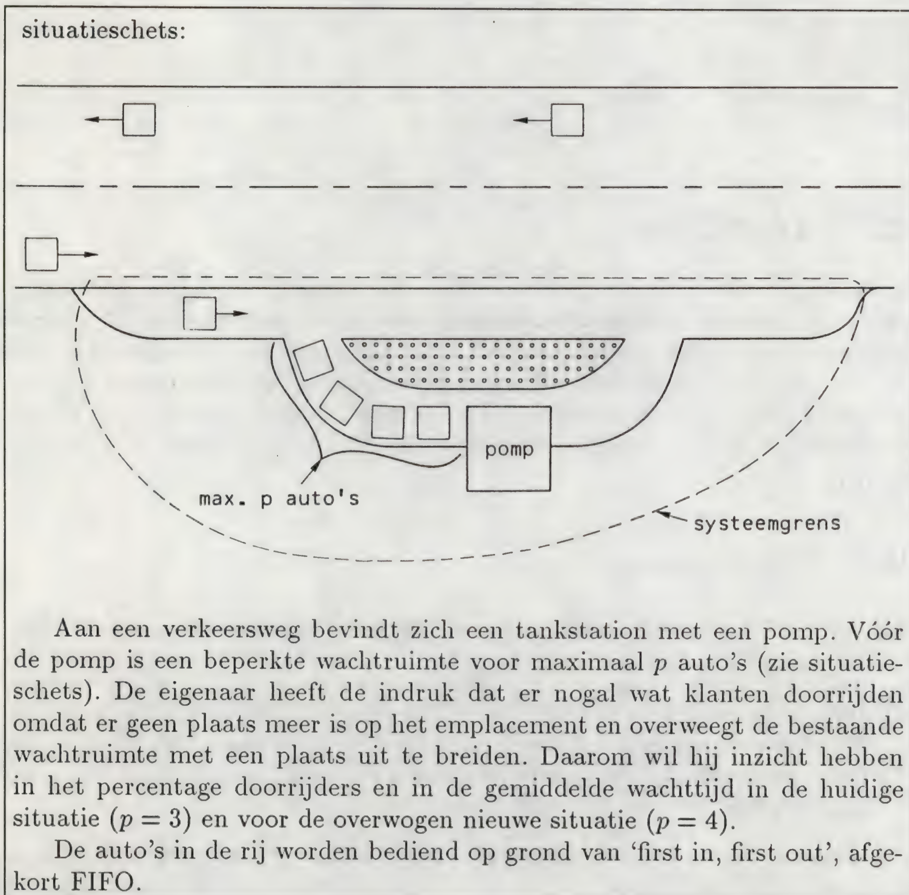
Een voor dit moment geschikte definitie van systeem is de volgende (In 't Veld 1984):

> *Een* **systeem** *is een, afhankelijk van het door de onderzoeker gestelde doel, binnen de totale werkelijkheid te onderscheiden* **verzameling elementen**. *Deze elementen hebben* **onderlinge relaties** *en (eventueel) relaties met andere elementen uit de totale werkelijkheid.*

In plaats van de term element wordt bij simulatietoepassingen vaak de term component gebruikt. In figuur 2.1 zijn een aantal voorbeelden van een systeem weergegeven. Daarvan wordt het tankstation in voorbeeld 2.1 nader toegelicht. Dit voorbeeld zal in dit boek stap voor stap worden uitgewerkt (een aantal van de hierbij voorkomende begrippen zijn ontleend aan Kerbosch et al. 1973).

Systeem	Componenten	Relaties
tankstation	auto's, pompen, wachtrij	de auto's staan in een wachtrij, de auto's tanken aan een pomp
supermarkt	klanten, kassa's, kassières, wachtrijen	de klanten worden door de kassières aan de kassa's geholpen
fabriek	orders, machines, werknemers, wachtrijen	de werknemers voeren met behulp van de machines de orders uit.

Figuur 2.1: Een drietal voorbeelden van een systeem



Aan een verkeersweg bevindt zich een tankstation met een pomp. Vóór de pomp is een beperkte wachtruimte voor maximaal p auto's (zie situatieschets). De eigenaar heeft de indruk dat er nogal wat klanten doorrijden omdat er geen plaats meer is op het emplacement en overweegt de bestaande wachtruimte met een plaats uit te breiden. Daarom wil hij inzicht hebben in het percentage doorrijders en in de gemiddelde wachttijd in de huidige situatie ($p = 3$) en voor de overwogen nieuwe situatie ($p = 4$).

De auto's in de rij worden bediend op grond van 'first in, first out', afgekort FIFO.

Voorbeeld 2.1: *Een tankstation met een beperkte wachtruimte*

2.2.1 Het karakteriseren van een systeem

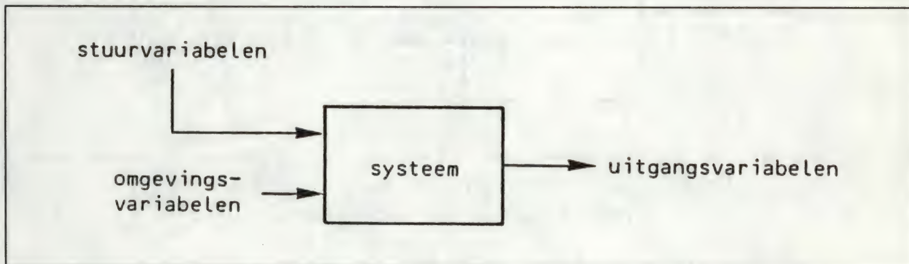
Voor het bestuderen van een systeem bestaan er twee benaderingsmethoden:

- vanuit het geheel (top-down),
- vanuit de elementen (bottum-up).

Bij de eerste methode, die kenmerkend is voor de zogenaamde 'systeembenadering', wordt een systeem in eerste instantie opgevat als een 'black-box' met de nadruk op de interactie met de omgeving. Bij de tweede methode gaat men uit van de componenten van een systeem en de relaties tussen deze componenten (zie definitie 'systeem'). Bij simulatietoepassingen zijn beide benaderingsmethoden van belang.

2.2.2 De systeembenadering (top-down)

In figuur 2.2 is een systeem in relatie tot zijn omgeving getekend. Uit de doelstelling van het onderzoek dient afgeleid te worden welk deel van de werkelijkheid men als systeem wenst te beschouwen; dit is het vaststellen van de zogenaamde systeemgrens. De van belang zijnde variabelen of grootheden die eveneens uit de doelstelling van het onderzoek volgen, kunnen in drie groepen worden ingedeeld.



Figuur 2.2: Een systeem in relatie tot zijn omgeving

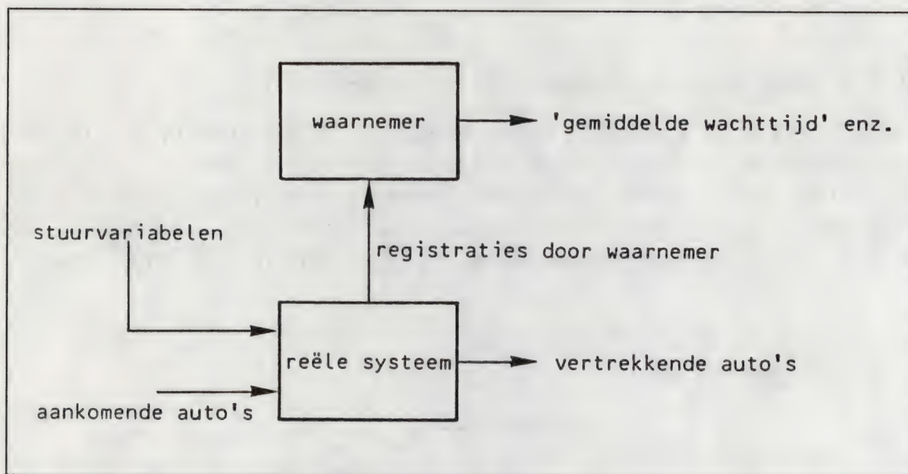
De *uitgangsvariabelen*, ook wel afhankelijke of endogene variabelen genoemd, hebben betrekking op de functie van het systeem in zijn omgeving. Het vervullen van die functie in de omgeving is het doel van het systeem.

De *ingangsvariabelen*, ook wel onafhankelijke of exogene variabelen genoemd, kunnen worden onderscheiden in *stuurvariabelen* of beslissingsvariabelen welke gebruikt worden om het systeem bewust te beïnvloeden, en in *omgevingsvariabelen* die als een gegeven externe invloed op het systeem aanwezig zijn.

Met betrekking tot voorbeeld 2.1 omsluit de systeemgrens de pomp, de eventueel aanwezige auto's (zowel de bediende als de in de wachtrij staande auto's) en de uitvoeg- en invoegstrook. Daardoor zal een aankomende auto die een volle wachtrij ziet en doorrijdt voor een korte tijd deel uitmaken van het systeem. De aankomsttijden en de bedieningstijden van de aankomende

auto's, vormen hier de omgevingsvariabelen. De stuurvariabele bestaat uit de grootte p van de wachtruimte. De uitgangsvariabelen bestaan uit, enerzijds de vertrekkende auto's en anderzijds uit de gewenste informatie: 'het percentage doorrijders' en de 'gemiddelde wachttijd'.

Om een duidelijk onderscheid aan te brengen tussen het reëele systeem met zijn output in de vorm van vertrekkende auto's en de aan het reële systeem te ontlenen gewenste informatie ('gemiddelde wachttijd' etc.) delen we het systeem op in twee delen. Zie figuur 2.3. De *waarnemer* moet er voor zorgen dat de gewenste informatie ('gemiddelde wachttijd' etc.) wordt verkregen. Hiertoe verricht hij interne registraties.



Figuur 2.3: Het tankstation voorgesteld als een reëel systeem met waarnemer

Via de bottom-up aanpak worden hierna de componenten en de structuur van het reële systeem bepaald.

2.2.3 De componenten en hun eigenschappen (bottom-up)

De componenten (ook wel elementen, objecten of entiteiten genoemd) zijn de kleinste delen van het reële systeem die de onderzoeker wenst te beschouwen. De componenten van een systeem kan men op grond van gelijksoortig gedrag indelen in *componentklassen*. Dit zal samenhangen met de eigenschappen of attributen die men aan de component wil onderscheiden.

Bij het tankstation gedragen alle auto's zich op soortgelijke wijze. De auto's vormen een componentklasse. Hetzelfde geldt voor de pomp(en). Bij dit voorbeeld zou naast een klasse van personenauto's bijvoorbeeld nog een klasse van vrachtauto's binnen het tankstationsysteem kunnen worden onderscheiden, indien het gedrag van vrachtauto's binnen het tankstation essentieel anders zou zijn dan het gedrag van personenauto's.

Een component wordt nader gespecificeerd door de waarden van zijn attributen. Bij de auto's zijn dit het moment van aankomst en de bedieningstijd, voor de wachtrij de lengte van de wachtrij, en voor de pomp het al dan niet bezet zijn hiervan.

Componenten die tot een klasse behoren, verschillen alleen ten aanzien van hun attribuutwaarden.

Men onderscheidt *tijdelijke* en *permanente* componenten. Een component heet tijdelijk als deze niet gedurende het gehele tijdsinterval waarin het reële systeem wordt beschouwd daarvan deel uitmaakt. In voorbeeld 2.1 zijn de auto's tijdelijke componenten en de pomp en de wachtrij permanente componenten. Soms kan men permanente componentklassen nog verder opsplitsen in beweeglijke en vaste componentklassen. Bijvoorbeeld binnen een fabriek kunnen vorkheftrucks beweeglijke en machines vaste componentklassen zijn.

2.2.4 Relaties tussen componenten; toestand van een systeem, event en proces

De verzameling relaties tussen de componenten van een systeem noemt men de **structuur** *van het systeem.*

(De in de definitie van systeem genoemde relaties met componenten van buiten het systeem, worden gevat onder de reeds genoemde ingangs- en uitgangsvariabelen.)

De elementen kunnen elkaar zo beïnvloeden. Dit betekent dat een verandering van de waarde van een attribuut van een element een verandering in waarde van attributen van andere elementen tot gevolg kan hebben en eventueel omgekeerd. Bijvoorbeeld het niet meer bezet zijn van de pomp kan tot gevolg hebben dat de lengte van de wachtrij wordt verlaagd met één. De toestand van het systeem zal zich wijzigen.

De **toestand** *van een systeem op een bepaald tijdstip wordt volledig beschreven door de waarden van de attributen van de componenten van het systeem op dat moment.*

De attributen, waarvan de waarden in de loop der tijd kunnen veranderen, worden *toestandsvariabelen* genoemd. In het voorbeeld 2.1 zijn het al of niet bezet zijn van de benzinepomp en het aantal auto's in de wachtrij de toestandsvariabelen. De toestand van een systeem op een gegeven ogenblik is een gevolg van voorafgaande gebeurtenissen of events.

Het moment waarop de toestand van een systeem verandert, noemt men een **eventtijdstip**.

Een **event** *vormt de aanleiding tot een of meer op hetzelfde eventtijdstip plaatsvindende activiteiten die de toestand van het systeem doen veranderen.*

De events kan men indelen in klassen op grond van het gelijksoortig zijn van de erbij behorende activiteiten en toestandsveranderingen die daarbij plaats kunnen vinden. Bij het tankstation zijn de aankomsten van de auto's events van dezelfde klasse, immers al deze events kunnen aanleiding geven tot een soortgelijke verandering van het systeem, namelijk verlenging van de wachtrij c.q. bezetting van de pomp. Het beëindigen van het tanken van een auto is een event van een andere klasse, immers dit event geeft aanleiding tot het vrijkomen van de pomp c.q. de bezetting van de pomp door de eerste auto uit de wachtrij.

De toestand van het systeem wordt steeds veranderd door activiteiten. De feitelijke uitvoering van de activiteiten vindt plaats door de componenten.

De door een bepaalde component in de loop der tijd uit te voeren activiteiten wordt een **proces** *genoemd.*

In voorbeeld 2.1 vormen de volgende 'activiteiten' een voorbeeld van een proces van een auto: aankomst auto op tijdstip t1, wachten op het emplacement (wachtrij) gedurende t2, het tanken gedurende t3 tijdseenheden en vervolgens het vertrek.

2.2.5 Soorten systemen

In het algemeen kunnen de volgende soorten systemen onderscheiden worden:

a. *Statische* en *dynamische* systemen.
   Bij dynamische systemen is minstens één toestandsvariabele een functie van de tijd. Bij statische systemen is dit niet het geval.

b. *Deterministische* en *stochastische* systemen.
   Bij stochastische systemen is er tenminste één stochastische variabele, die het gedrag van het systeem beïnvloedt. Bij deterministische systemen zijn er geen stochastische variabelen.

c. *Discrete* en *continue* systemen.
   Een systeem is een *discreet systeem* wanneer alle toestandsvariabelen van het systeem discrete functies van de tijd zijn. De toestand van zo'n systeem verandert slechts op een bepaald aantal momenten. Tussen twee opeenvolgende eventtijdstippen verandert de toestand van het systeem niet. De veranderingen zelf verlopen tijdloos. De in figuur 2.1 genoemde voorbeelden zijn voorbeelden van discrete systemen. De componenten van een discreet systeem doorlopen scherp gescheiden fasen. Wanneer een component overgaat in een volgende fase, verandert de toestand van het systeem. Dit zijn ook de enige momenten waarop de toestand verandert. Het systeem gaat als het ware schoksgewijs van de ene toestand over in de volgende toestand. Bij een tankstation bijvoorbeeld worden de toestandsveranderingen veroorzaakt door aankomst respektievelijk vertrek van een auto. De auto's doorlopen de scherp gescheiden fasen 'wachtend'

en 'in bediening', de pomp doorloopt de fasen 'bezig' en 'niet bezig' (zie ook figuur 2.5).

Men spreekt van een *continu systeem* wanneer de toestand van het systeem in de tijd continu verandert. Er is dan tenminste één toestandsvariabele die een continue functie van de tijd is. Thermische processen, de groei van planten en analoge elektrische schakelingen zijn voorbeelden van continue systemen. Soms is het door een geschikte keuze van het model toch mogelijk een continu systeem discreet te behandelen, bijvoorbeeld een continu productieproces kan soms worden voorgesteld als een regelmatige serie kleine 'batches'.

Wij zullen ons in het vervolg slechts bezighouden met dynamische, stochastische, discrete systemen, kortweg aan te duiden met 'systeem'.

2.3 Modellen

We zullen hier onder een model verstaan:

Een **model** *is een vereenvoudigde afbeelding van een reëel systeem waarbij alleen die componenten en relaties worden beschouwd die van belang zijn binnen het kader van de gegeven probleemstelling.*

Een model is dus een vereenvoudigde afbeelding van de realiteit, maar komt qua structuur wel overeen met die realiteit. Bij deze afbeelding kunnen afbeeldfouten worden gemaakt.

Voorbeeld 2.2: *Het model van het tankstation.*

Uitgaande van de in voorbeeld 2.1 gegeven omschrijving van het tankstation zijn de volgende componentklassen te onderscheiden:

Vaste componentklassen:

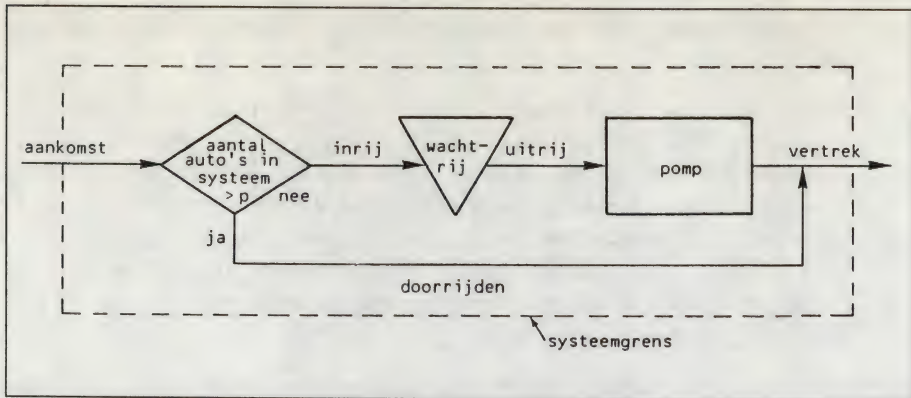
- pomp,
- wachtrij.

Tijdelijke componentklasse:

- auto.

Om tot de structuur van het model te komen zullen we tussen de componentklassen de relaties moeten aangeven. Dit kan door het proces, dat doorlopen wordt door de tijdelijke (beweeglijke) componenten, te beschouwen. We nemen hierbij aan dat een aankomende auto ook bij een leeg systeem via de wachtrij plaats neemt bij de pomp.

De structuur van het model is in figuur 2.4 weergegeven.



Figuur 2.4: Een grafische weergave van het conceptueel model van het tankstation

De betekenis van de gebruikte symbolen is:

rechthoek	:	permanente component (behalve wachtrij)
driehoek	:	wachtrij
ruit	:	keuze
pijl	:	bewegingsrichting van tijdelijke (beweeglijke) component.

In figuur 2.4 is aangegeven dat de tijdelijke component betrokken kan zijn bij de activiteiten: aankomst, doorrijden, inrij, wachten, uitrij, tanken en vertrek. De wachtrij is betrokken bij inrij, wachten, uitrij en de pomp bij uitrij, tanken en vertrek.

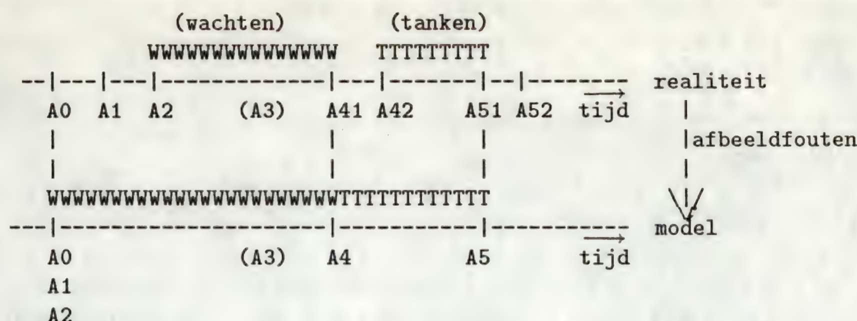
Voor een aantal activiteiten kan worden aangegeven dat ze op een zeker tijdstip plaatsvinden (aankomst, doorrijden, inrij, uitrij en vertrek), andere activiteiten gaan gepaard met een zekere tijdsduur (wachten en tanken).

Aangezien bij discrete modellen alle toestandsveranderingen momentaan plaatsvinden zullen we alle activiteiten die men in het reële systeem wil onderscheiden moeten beschrijven middels activiteiten die tijdloos zijn. Dit wil hier zeggen dat het wachten wordt beschreven als een interval dat start met 'inrij' en waarvan de beëindiging wordt aangeduid met 'uitrij'. Evenzo geldt voor het tanken dat het start met 'uitrij' en dat de beëindiging wordt aangeduid met 'vertrek'.

De samenhang tussen de hiervoor genoemde activiteiten in de tijd is aangegeven in figuur 2.5.

In figuur 2.5 is tevens aangeduid welke modelaannames we zullen gaan maken om binnen het kader van de doelstellingen tot een eenvoudig doch realistisch model te komen. Deze aannames (*afbeeldfouten*) zijn:

- De tijd vanaf het aankomstmoment in het systeem tot het tijdstip dat de auto doorrijdt (bij volle wachtrij) of tot het tijdstip dat de auto in de wachtrij heeft plaatsgenomen wordt verwaarloosd (het eventuele wachten begint bij A0),




```
 A0: AANKOMST in systeem
 A1: DOORRIJDEN
 A2: INRIJ
(A3: SCHUIFDOOR in wachtrij t.g.v. het vertrek van een voorganger)
A41: auto uit rij
A42: start tanken
 A4: UITRIJ (= A41 + A42)
A51: tanken klaar
A52: vertrek
 A5: VERTREK (= A51 + A52)
```

Figuur 2.5: Onderkende activiteiten in realiteit en model

- De tijd nodig om de wachtrij te verlaten en naar de pomp te rijden wordt in de bedieningstijd verdisconteerd (de periode A41-A42 wordt ook als tanktijd opgevat),
- Het vertrek van de auto valt samen met het moment dat de bediening klaar is (A52 valt samen met A51).

Modelaannames met betrekking tot de keuze van verdelingen voor tussen-aankomsttijden en bedieningstijden van de auto's worden in hoofdstuk 4 (statistische aspecten) behandeld. Deze afbeeldfouten hebben namelijk betrekking op het kwantitatieve model.

Wanneer men tot iedere prijs het maken van afbeeldfouten wenst te voorkomen, kan het model zeer gecompliceerd worden. Men zal steeds schipperen tussen aan de ene kant 'de mate waarin het model lijkt op de werkelijkheid' en aan de andere kant de doorzichtigheid ervan. Men dient daarbij na te gaan welke afbeeldfouten worden gemaakt en in hoeverre deze afbeeldfouten de resultaten van het onderzoek beïnvloeden.

We gaan nu het model van het tankstation detailleren door de componentklassen nader te specificeren:

De toestandsvariabelen zijn in dit voorbeeld de attributen van de vaste componentklassen: RIJLENGTE en POMPVRIJ.

Om het gedrag van het systeem als functie van de tijd te kunnen beschrijven wordt een variabele TIME ingevoerd, die de systeemtijd weergeeft.

Voor het verkrijgen van de gewenste output (zie figuur 2.3) worden de volgende hulpvariabelen gedefinieerd:

NAANKOMST	: aantal aangekomen auto's tot nu toe
NBEDIEND	: aantal bediende auto's tot nu toe
NDOORRIJDEN	: aantal doorgereden auto's tot nu toe
WACHTTIJDi	: wachttijd AUTOi

De in figuur 2.5 gedefinieerde activiteitsklassen geven aanleiding tot de volgende veranderingen van de waarden van variabelen. (N.B. de toestandsvariabelen zijn cursief gedrukt):

activiteitsklassen		verandering van variabelen	
AANKOMST	: auto komt systeem binnen	NAANKOMST	:= NAANKOMST + 1
DOORRIJDEN	: auto rijdt door	NDOORRIJDEN	:= NDOORRIJDEN + 1
INRIJ	: auto in rij	*RIJLENGTE*	:= *RIJLENGTE + 1*
SCHUIFDOOR	: auto schuift door in rij		
UITRIJ	: auto uit rij + start tanken	*RIJLENGTE*	:= *RIJLENGTE − 1*
		POMPVRIJ	:= FALSE
		WACHTTIJDi	:= TIME-AANKOMSTTIJDi
VERTREK	: tanken klaar + vertrek	*POMPVRIJ*	:= TRUE
		NBEDIEND	:= NBEDIEND + 1

In het bovenstaande staat AANKOMSTTIJDi voor de AANKOMSTTIJD van de i-de auto. Voor het kunnen uitvoeren van een daadwerkelijke simulatie moeten de activiteiten, die op hun beurt de toestandsveranderingen veroorzaken, nog in de tijd worden geordend.

De hiervoor behandelde activiteitsklassen kunnen daarbij op verschillende manieren worden gegroepeerd. Afhankelijk van de wijze van groeperen ontstaan de volgende drie beschrijvingswijzen:

- activiteitenbeschrijving (afzonderlijke beschrijving van elke activiteit met bijbehorende voorwaarden)
- eventbeschrijving (groepering activiteiten rondom tijdstip van optreden event)
- procesbeschrijving (groepering activiteiten rondom componenten)

Deze beschrijvingswijzen zullen in de volgende paragrafen behandeld worden.

2.4 De activiteitenbeschrijving

Bij het gebruik van een activiteitenbeschrijving wordt het systeem beschreven door een beschrijving te geven van alle soorten activiteiten (handelingen, acties) die in het systeem voor kunnen komen. Voor iedere activiteitsklasse dienen de voorwaarden te worden gegeven waaraan moet zijn voldaan om die activiteit uit te kunnen voeren. Deze voorwaarden kunnen afhankelijk zijn van de systeemtijd.

Bij het tankstationprobleem kunnen de volgende voorwaarden aan de hiervoor gedefinieerde activiteitsklassen worden toegevoegd:

IF TIME=AANKOMSTTIJD		THEN AANKOMST
IF TIME=AANKOMSTTIJD	en RIJLENGTE = p	THEN DOORRIJDEN
IF TIME=AANKOMSTTIJD	en RIJLENGTE < p	THEN INRIJ
IF RIJLENGTE > 0	en POMPVRIJ=TRUE	THEN UITRIJ + SCHUIFDOOR
IF TIME=eindtijd actuele bediening		THEN VERTREK

Met betrekking tot het bovenstaande kunnen we opmerken dat de tijdvoorwaarde van de eerste drie activiteitsklassen identiek is. Deze activiteitsklassen kunnen daarom gezamenlijk worden beschreven. Van de activiteitsklassen UITRIJ en SCHUIFDOOR geldt dat de activiteit SCHUIFDOOR steeds in combinatie met UITRIJ voorkomt en deze kunnen daarom ook als een geheel worden beschreven. Zodoende kunnen we ons hier in wezen beperken tot de volgende (samengestelde) activiteitsklassen:

1. Plaats een auto in de wachtrij.
   Hier is de voorwaarde dat door de systeemtijd wordt aangegeven dat het aankomstmoment van een auto bereikt is.
   De auto wordt aan de wachtrij toegevoegd, tenzij de wachtrij daardoor meer dan p auto's zou gaan bevatten. Door de aankomst van de auto start er een pauzeduur tussen twee aankomsten.
   (De duur van die pauze wordt bijvoorbeeld bepaald door middel van een trekking uit de verdeling van tussenaankomsttijden. Hierdoor wordt een volgend eventtijdstip verkregen waarop er weer een auto zal arriveren).

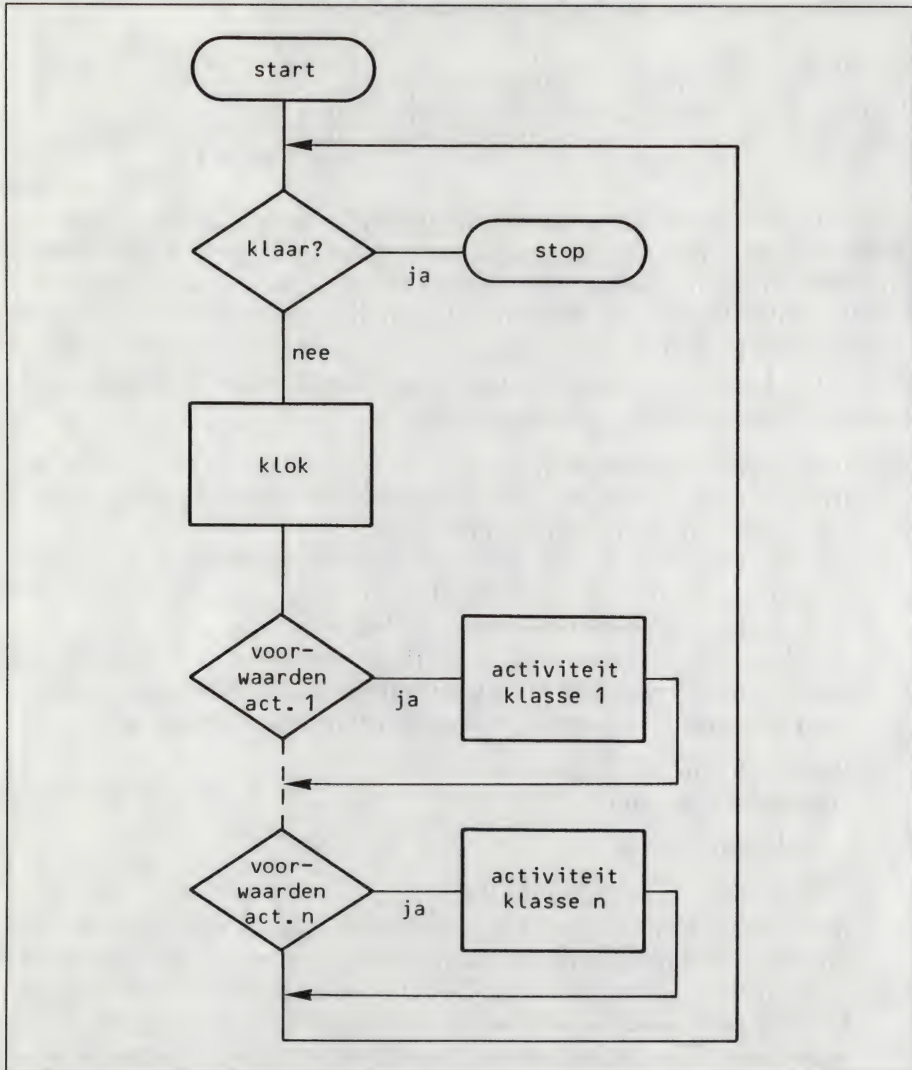
2. Plaats een auto bij de pomp.
   De voorwaarden zijn:

   - de pomp is vrij,
   - er is een auto in de wachtrij.

   Zodra een dergelijke situatie optreedt, zal de voorste auto in de wachtrij bij de pomp worden geplaatst, waardoor de pomp wordt bezet. Het bedieningsproces gaat dan van start. (De duur van die bediening wordt bijvoorbeeld bepaald door middel van een trekking uit de verdeling van bedieningstijden. Hierdoor wordt een volgend eventtijdstip verkregen waarop een bediening klaar komt.)
   Indien er nog auto's in de wachtrij zijn overgebleven schuiven deze een plaats in de wachtrij door.

3. Verwijder een auto uit het systeem.
   De voorwaarde is nu dat de systeemtijd het moment aangeeft, dat een pompbediening is klaargekomen. De auto bij de pomp zal dan het systeem verlaten, waardoor de pomp weer vrij komt.

Bij simulatie van het hier beschreven model worden de voorwaarden van alle activiteiten achtereenvolgens gecontroleerd (zie figuur 2.6). Wordt aan alle voorwaarden voor het starten van een activiteit voldaan, dan wordt die activiteit uitgevoerd.



Figuur 2.6: Stroomschema activiteitenbeschrijving (rangschikking activiteiten in volgorde van optreden)

Kan geen activiteit meer worden uitgevoerd, dan zal de systeemtijd worden verhoogd tot het eerstvolgende eventtijdstip. Hiervoor zorgt de 'klok' in figuur 2.6. (De werking van de klok wordt nader toegelicht in 2.5).

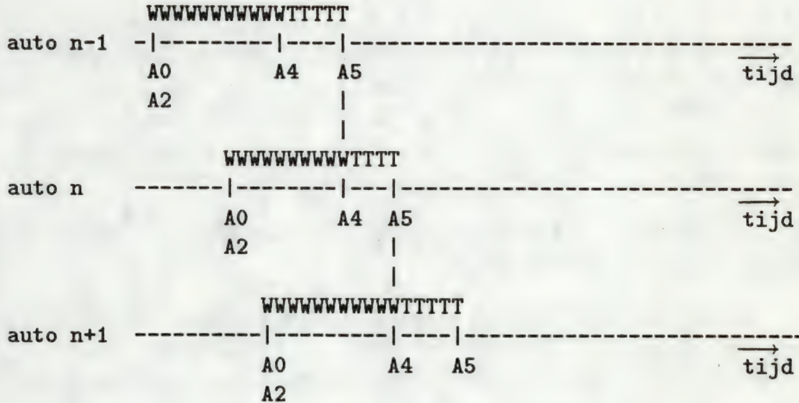
Omdat bij elk eventtijdstip steeds alle voorwaarden worden gecontroleerd is de methode niet bijzonder efficiënt. Het voordeel is echter, dat in een programma-onderdeel waar de beschrijving van een activiteit aan de orde komt ook alle bijzonderheden van die activiteit zijn te vinden.

2.5 De eventbeschrijving

Onder een *eventbeschrijving* verstaan we:

De beschrijving van alle (activiteiten en bijbehorende) toestandsveranderingen die op een eventtijdstip kunnen plaatsvinden.

Bij een eventbeschrijving gaat het uiteindelijk om de er door veroorzaakte toestandsveranderingen. Om de relatie met de andere beschrijvingswijzen duidelijk aan te kunnen geven zullen we als tussenstap de activiteitsklassen beschouwen. Door figuur 2.5 uit te breiden voor meerdere auto's zien we dat activiteit A4 van auto n samen kan vallen met A5 van auto n-1 (zie figuur 2.7).



Figuur 2.7: De activiteiten van opeenvolgende tijdelijke componenten

Indien bij aankomst van auto n het systeem leeg is zal voor auto n A4 samenvallen met A0. We zien dus dat in het geval van het tankstation de activiteiten A0, A1, A2 en A4 op hetzelfde eventtijdstip kunnen plaatsvinden. Het bij A0 behorende event duiden we aan met 'Aankomst' omdat activiteit A0 steeds de oorzaak is van het eventueel optreden van de overige activiteiten.

Evenzo is de oorzaak van het activiteitencluster A5, A4 de activiteit A5. Dit event wordt gemakshalve aangeduid met 'Vertrek'.

Voor de events Aankomst (= A) en Vertek (= V) van een auto luiden de eventbeschrijvingen dan:

A (Aankomst) : Indien de pomp vrij is wordt deze bezet: anders wordt de wachtrij één langer, tenzij de wachtrij daardoor langer wordt dan p auto's, dan zal de auto doorrijden.

V (Vertrek) : De pomp komt vrij. Indien er nog auto's in de wachtrij staan, dan gaat de voorste auto uit de wachtrij naar de pomp. De eventueel aanwezige overige auto's schuiven een plaats op in de wachtrij. Indien er geen auto's in de wachtrij staan, gebeurt er niets en is het wachten op een volgende aankomst.

De eventbeschrijvingen kunnen we als volgt omschrijven. (De verandering van de waarden van toestandsvariabelen staat tussen haakjes toegevoegd):

```
A : AANKOMST;
    IF RIJLENGTE = p THEN DOORRIJDEN ELSE INRIJ (RIJLENGTE:=RIJLENGTE+1)
    IF POMPVRIJ=TRUE THEN UITRIJ                (RIJLENGTE:=RIJLENGTE-1)
                                                POMPVRIJ:=FALSE)

V : VERTREK                                     (POMPVRIJ:=TRUE);
    IF RIJLENGTE > 0 THEN UITRIJ                (RIJLENGTE:=RIJLENGTE-1
                                                POMPVRIJ:=FALSE)
    IF RIJLENGTE > 0 THEN SCHUIFDOOR.
```

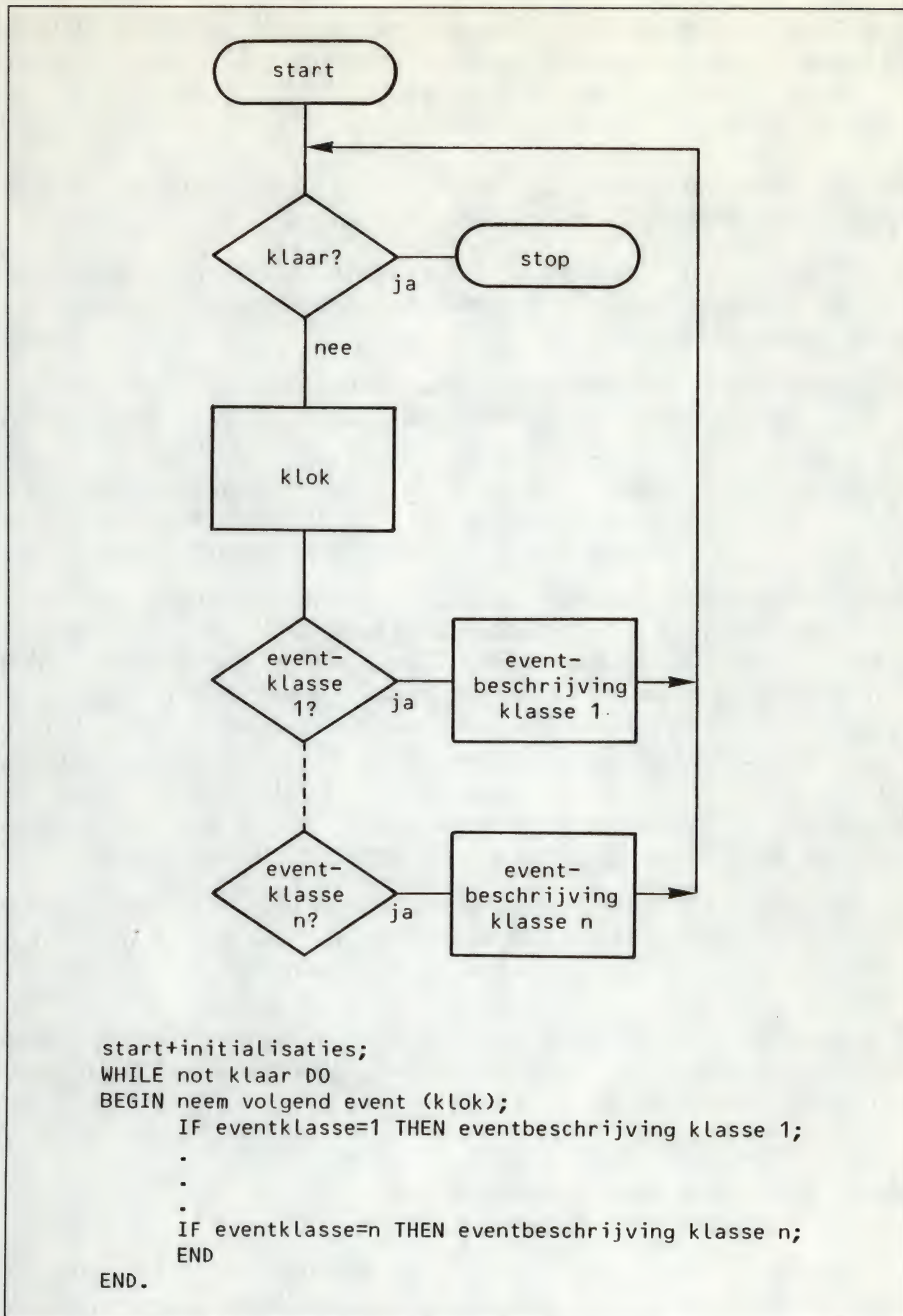
Met de eventbeschrijvingen is de werking van het systeem nog niet volledig vastgelegd. Er is namelijk nog niet beschreven op welke tijdstippen de diverse events plaatsvinden. Wanneer we zouden beschikken over een lijst van alle eventnotities (d.w.z. het eventtijdstip en bijbehorende eventklasse) dan zou een beschrijving van het systeem volledig gemaakt kunnen worden met het volgende voorschrift:

Werk in chronologische volgorde de lijst af en voer de in de eventbeschrijving voorgeschreven activiteiten uit.

Het vooraf ontwerpen van een volledige lijst van eventnotities is omslachtig en vaak onmogelijk.

Op ieder moment zijn we echter slechts geïnteresseerd in het eerstvolgende event (zie ook handsimulatie hoofdstuk 3). Het is daardoor mogelijk de lijst van eventnotities beperkt te houden. We moeten er alleen voor zorgen dat deze lijst altijd tenminste een eventnotitie (het eerstvolgende) bevat. We zullen hiertoe het volgende mechanisme hanteren, dat er voor zorgt dat de lijst van eerstkomende eventnotities niet groter dan noodzakelijk is.

1. Klokmechanisme: Selecteer de eerstvolgende eventnotitie. Dit kan in ons geval zowel op een aankomst als een vertrek betrekking hebben.




```
start+initialisaties;
WHILE not klaar DO
BEGIN neem volgend event (klok);
      IF eventklasse=1 THEN eventbeschrijving klasse 1;
      .
      .
      .
      IF eventklasse=n THEN eventbeschrijving klasse n;
      END
END.
```

Figuur 2.8: Stroomschema en pseudocode eventbeschrijving

2. Werk voor het onder punt 1. geselecteerde event alle activiteiten af. Voeg indien nodig een nieuwe eventnotitie aan de lijst toe. Verwijder de onder punt 1 genoemde eventnotitie uit de lijst.
3. Ga naar 1.

Het bijbehorende stroomschema is weergegeven in figuur 2.8. Tevens is in deze figuur de eventbeschrijving in pseudocode opgenomen.

Het bepalen van toekomstige eventtijdstippen en het vormen van de desbetreffende notities dient in de eventbeschrijving te worden opgenomen. In het geval van het tankstation:

- Bij aankomst : Bepaal het volgende aankomsttijdstip bijvoorbeeld met behulp van een trekking uit een verdeling van de tussenaankomsttijden.
- Bij vertrek : Indien er nog een wachtende auto aanwezig is bepaal dan het vertrektijdstip van deze auto bijvoorbeeld door een trekking uit een verdeling van de bedieningstijden.

Soms blijken eventbeschrijvingen een hoge mate van overeenkomst te vertonen. Bijvoorbeeld stel dat er eventbeschrijvingen zijn van aankomsten van een order van soort 1 en van soort 2 die slechts in geringe mate verschillen. Men kan dan de aankomsten van de twee ordersoorten onderbrengen in één eventklasse: de aankomst van een order. Om toch een onderscheid tussen de twee typen orders te kunnen maken moet deze eventbeschrijving wel een parameter ordersoort hebben. Ditzelfde kan mogelijk ook voor de eventbeschrijving van het beschikbaar komen van machines. Hierdoor kan men soms zelfs voor een vrij ingewikkeld systeem volstaan met een gering aantal eventtypen.

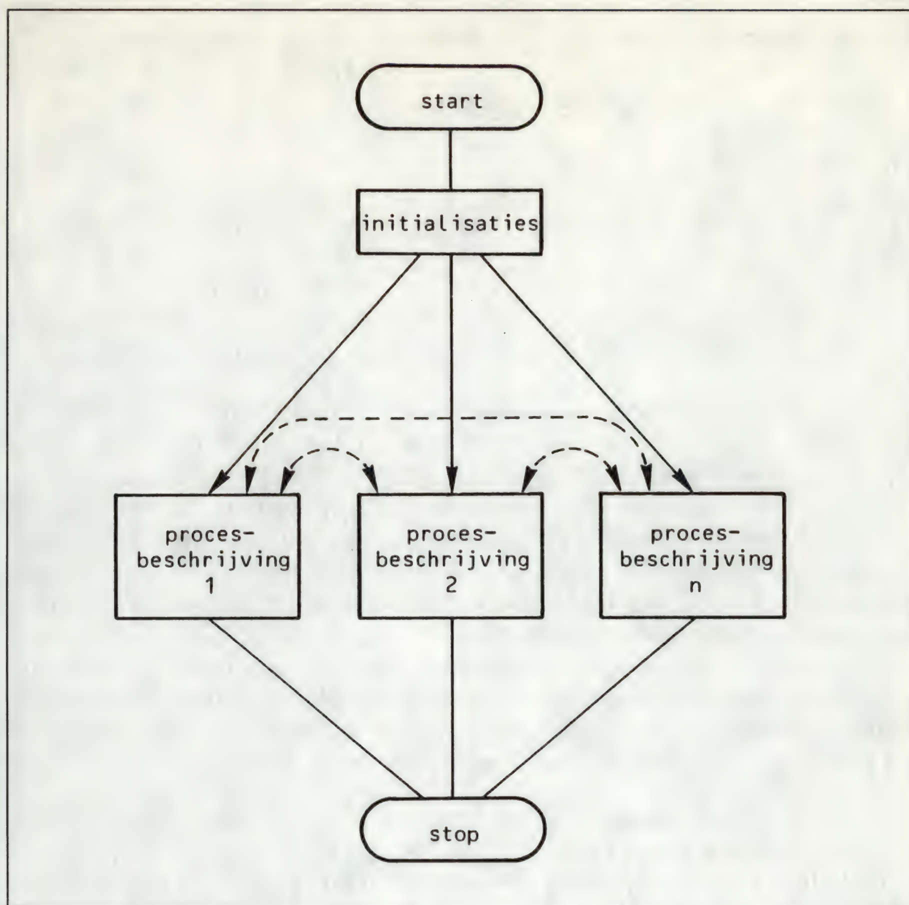
Als we de hier gegeven eventbeschrijving vergelijken met de activiteitenbeschrijving dan zien we dat de eventbeschrijving uit twee klassen en de activiteitenbeschrijving uit drie klassen bestaat. Hoe kan dat?

Dit komt doordat de afzonderlijk omschreven activiteit 'plaats een auto bij de pomp' (= auto uit rij + schuifdoor + start tanken) bij een eventbeschouwing altijd samen valt met een 'aankomst' van een nieuwe auto of het 'vertrek' van een vorige auto (zie ook figuur 2.7).

2.6 De procesbeschrijving

Een **procesbeschrijving** *van een componentklasse is de beschrijving van de activiteiten die een component van die klasse kan uitvoeren en van de daarbij behorende interacties met de overige componentklassen.*

In figuur 2.9 zijn verschillende procesbeschrijvingen met hun onderlinge interacties (aangeduid met stippellijnen) globaal weergegeven. Het klokmechanisme is hier 'verdeeld' over de verschillende procesbeschrijvingen.



Figuur 2.9: Het globale stroomschema volgens de procesbeschrijving. De stippellijnen symboliseren de interacties tussen de procesbeschrijvingen.

Het totale aantal activiteiten dat door alle componenten tesamen (het gehele systeem) wordt uitgevoerd veroorzaakt de toestandsveranderingen. Deze toestandsveranderingen mogen uiteraard niet afhangen van de gehanteerde beschrijvingswijze. Dit wil hier zeggen dat elk van de onderkende activiteiten eenduidig moet worden toegewezen aan een component die voor de uitvoering ervan zorgdraagt. Aan welke componentklasse een activiteitsklasse wordt toegewezen is hierbij in principe niet van belang. Echter om een modulaire structuur te bevorderen (vooral van belang bij implementatie) wordt een activiteitsklasse bij voorkeur toegewezen aan die componentklasse, die als attributen toestandsvariabelen bevat die door de activiteitsklasse gewijzigd worden.

Met betrekking tot voorbeeld 2.1 kunnen de onderscheiden activiteitsklassen worden toegewezen aan de componentklassen zoals bijvoorbeeld weergegeven in figuur 2.10

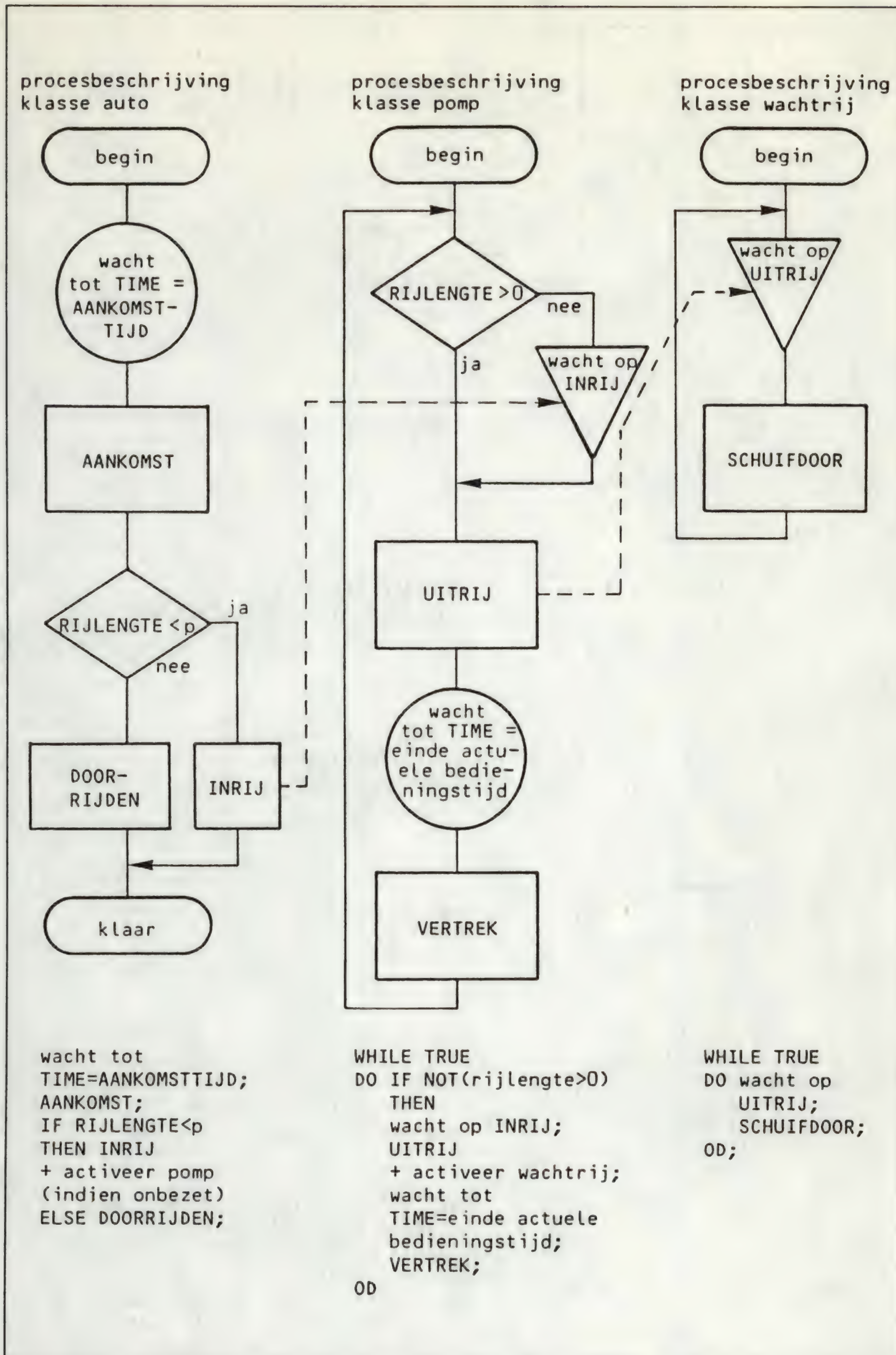
Activiteitsklassen	Componentenklassen			Verandering toestandsvariabele
	auto	wachtrij	pomp	
AANKOMST	X			
DOORRIJDEN	X			
INRIJ	X			RIJLENGTE
UITRIJ			X	POMPVRIJ
SCHUIFDOOR		X		RIJLENGTE
VERTREK			X	POMPVRIJ

Figuur 2.10: Activiteitsklassen toegewezen aan componentklassen

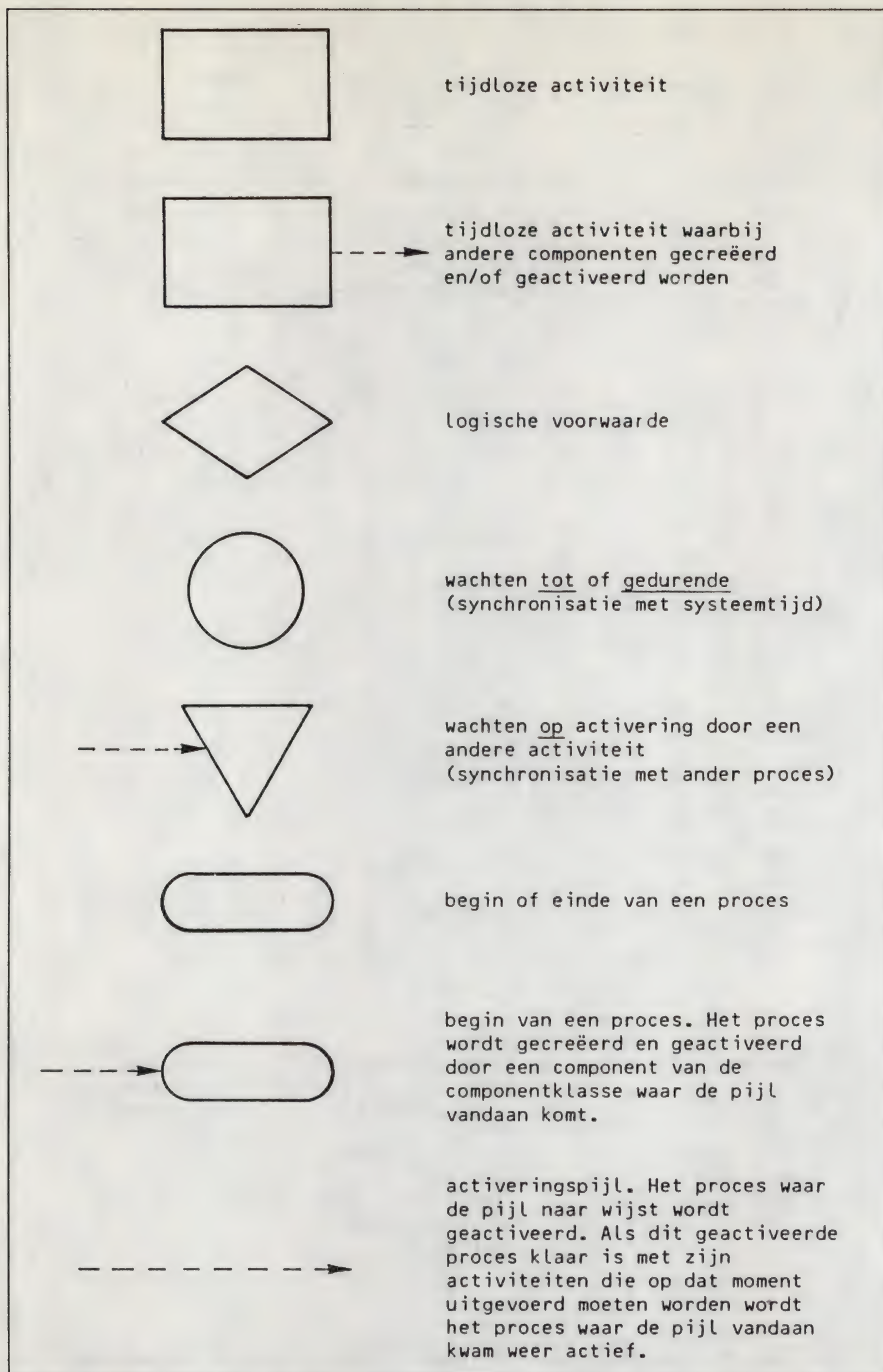
In deze figuur zijn tevens de toestandsvariabelen genoemd die door de genoemde activiteiten worden gewijzigd. Merk op dat deze toestandsvariabelen bij andere componentklassen (kunnen) behoren dan de verandering veroorzakende componentklasse. (Voorbeeld: INRIJ behorende bij auto wijzigt de toestandsvariabele RIJLENGTE van de wachtrij). Bovendien kan het voorkomen dat een componentklasse in het geheel geen activiteitklasse(n) toebedeeld gekregen heeft. Een dergelijke componentklasse blijft echter van belang als er toestandsvariabelen aan gekoppeld zijn.

We zullen nu het globale schema van figuur 2.9 gaan detailleren. De daar aangeduide procesbeschrijvingen zijn voor voorbeeld 2.1, uitgaande van figuur 2.10, uitgewerkt tot figuur 2.11. Deze figuur is daarbij als volgt tot stand gekomen (voor de betekenis van de symbolen zie figuur 2.12):

a. Teken per componentklasse de toegewezen activiteiten in chronologische volgorde van boven naar beneden,
b. Indien voor het optreden van een activiteit aan één of meer logische voorwaarde(n) voldaan moet worden, geef deze dan direct boven deze activiteit aan met behulp van een ruitsymbool,
c. Indien voor het optreden van een activiteit *gewacht* moet worden *op* het optreden van een activiteit die door een andere componentklasse wordt uitgevoerd (interactie), geef dit dan aan met een wachtsymbool (driehoekje),
d. Indien binnen een componentklasse voor het optreden van een activiteit *gewacht* moet worden *tot* er een zeker tijdstip binnen het systeem bereikt is (systeemtijd), geef dit dan aan d.m.v. een rondje (van een klok) boven deze activiteit,
e. Geef interacties tussen de componentklassen weer met stippellijnen, die starten in de activiteit die deze interactie veroorzaakt en eindigt in een wachtsymbool. Opgemerkt dient te worden dat er ook interacties voor kunnen komen tussen activiteiten van dezelfde componentklasse. Het betreft dan interacties tussen meerdere componenten van dezelfde klasse.



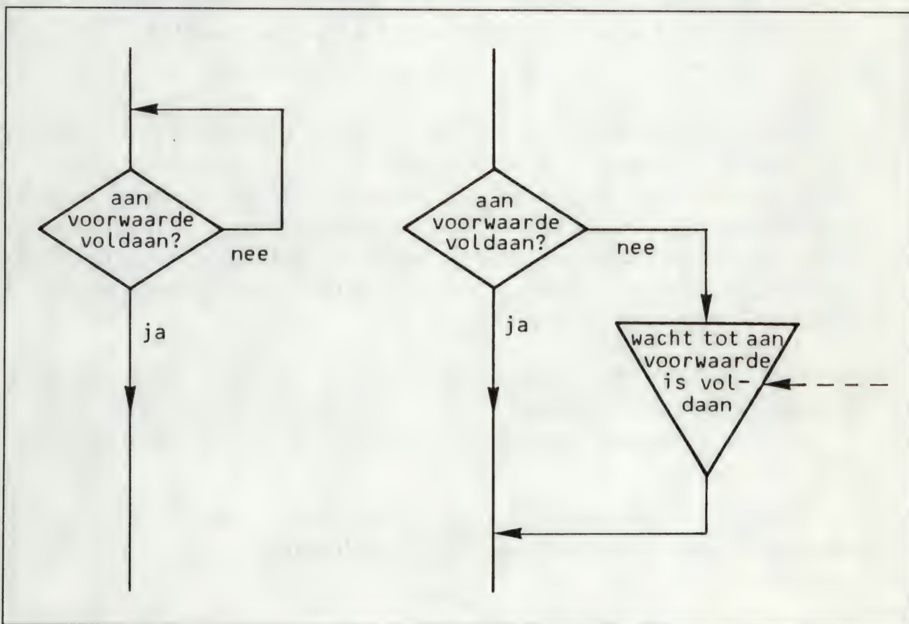
Figuur 2.11: De procesbeschrijvingen met bijbehorende pseudocode programma's voor het tankstation van voorbeeld 2.1



Figuur 2.12: Betekenis symbolen procesbeschrijvingen

Met betrekking tot de procesbeschrijvingen in figuur 2.11 merken we verder nog het volgende op:

a. Zoals reeds opgemerkt, wordt bij een procesbeschrijving elke activiteit precies één keer toegewezen aan een component die voor de uitvoering ervan zorgt draagt (en dus de toestandsverandering veroorzaakt). Daarom zien we deze activiteit niet voor een tweede keer terug bij de component die deze activiteit *passief* ondergaat. Bijvoorbeeld UITRIJ - wacht gedurende bediening - VERTREK wordt *ondergaan* door de auto maar is dus in diens procesbeschrijving niet zichtbaar.
b. De procesbeschrijvingen zijn op componentklasse nivo. Dit wil zeggen dat bij simulatie van een volledig model elke component van een klasse een eigen procesbeschrijving toebedeeld krijgt die beschouwd kan worden als zijnde een copie van de bijbehorende componentklasse. De hier gegeven interactie-stippellijnen tussen de klassen representeren interacties tussen afzonderlijke componenten van die klassen.
c. De procesbeschrijvingen zijn naast elkaar getekend om uit te drukken dat er componenten van verschillende componentklassen tegelijkertijd in het systeem aanwezig kunnen zijn.
d. Zoals de meeste software worden ook discrete simulatiemodellen (nog) geïmplementeerd op sequentiële computersystemen waarvan de processor eigenlijk maar één ding tegelijk kan doen. Om de procesbeschrijvingen te implementeren wordt gebruik gemaakt van *quasi-parallelliteit*.



Figuur 2.13: In de rechter figuur vindt de de-activatie plaats totdat aan de voorwaarde is voldaan.

Dit betekent dat een proces alleen echt actief is als er echt een activiteit uit te voeren is. Dit zullen we proberen te verduidelijken aan de hand van figuur 2.13 (zie vorige pagina).
Stel dat niet aan een bepaalde voorwaarde is voldaan. In het linkerschema zal *voortdurend* worden getest net zo lang tot aan de voorwaarde voldaan is (de processor van de computer is hier continu mee bezig; het programma blijft 'hangen'). Willen we de sequentiële processor gedurende het 'hangen' voor een andere activiteit gebruiken dan kan dat volgens het rechterschema. Daar wordt één keer getest en als dan niet aan de voorwaarde is voldaan treedt een de-activering van het proces op.
In deze laatste situatie kan gedurende de wachtperiode van het ene proces de processor voor de verwerking van een ander proces worden gebruikt dat dan actief wordt. Aan de wachtperiode van het eerste proces wordt een eind gemaakt door een ander proces dat er voor gezorgd heeft dat aan de voorwaarde voldaan is. De verwerking van het eerste proces kan nu worden voortgezet.

e. De componenten van de tijdelijke componentklassen doorlopen hun procesbeschrijving een keer (procesbeschrijving ingesloten tussen 'begin' en 'klaar'); de componenten van de permanente componentklassen kunnen meerdere malen gedurende de hele simulatietijd doorlopen worden (wel 'begin', geen 'klaar'),
f. Merk op dat indien een component na het uitvoeren van een activiteit niet direct een volgende activiteit kan uitvoeren, deze component als niet-actief (gedeactiveerd) beschouwd mag worden. Een gedeactiveerde component kan op twee manieren weer actief worden: of doordat een bepaald tijdsinterval verstreken is (rondje) of doordat een andere component hem 'activeert' (driehoek).
g. Merk op dat ook het verblijf van een tijdelijke component in de wachtrij niet zichtbaar is in de figuur. Dit komt doordat het 'verblijf' dat afgebakend wordt door de tijdloze activiteiten INRIJ en UITRIJ, aan twee verschillende componentklassen zijn toegewezen.
h. Merk tot slot op dat in de procesbeschrijving het aantal rondjes (= afstemmingen in de tijd) gelijk is aan het aantal eventklassen bij de eventbeschrijving (ga zelf na waarom).

We zien dat de procesbeschrijving overeenkomst vertoont met een min of meer intuïtieve beschrijving van voorbeeld 2.1. Het beschrijven van een systeem door middel van procesbeschrijvingen is een natuurlijke en voor de hand liggende wijze van beschrijven.

We hebben tot nu toe 'slechts' het procesbeschrijvingen gedeelte van figuur 2.9 gedetailleerd. Het resterende deel van deze figuur zullen we nu als volgt uitwerken (zie figuur 2.14):

a. In figuur 2.9 en 2.11 is niet aangegeven wanneer de simulatie zal eindigen. In het algemeen wordt de duur van een simulatie bepaald door het verwerkt zijn van een bepaald aantal tijdelijke componenten of door het

verstrijken van een vooraf vastgestelde tijdsduur. Voor het beëindigen van de simulatie en voor het creëren van de verschillende componenten zullen we een nieuwe componentklasse 'besturing' invoeren. De hiervoor genoemde stopcriteria worden in deze componentklasse weergegeven door middel van de reeds geïntroduceerde symbolen 'driehoek' en 'rondje'. De component 'besturing' in fig. 2.14 geeft precies aan in welke volgorde de componenten van de diverse klassen geactiveerd dienen te worden. Om een onderscheid aan te geven tussen de activiteiten van het reële systeem en de besturingsactiviteiten, zijn deze laatste met kleine letters aangeduid.

b. In figuur 2.11 is er van uitgegaan dat elke individuele auto 'achter de schermen' wacht tot zijn aankomstmoment is aangebroken. Dat zou inhouden dat voor elke auto de procesbeschrijving gestart moet worden aan het begin van de simulatie. Het is echter realistischer de tijdelijke componenten te creëren en te starten op het moment van hun aankomst in het systeem. Om deze reden wordt de componentklasse 'aankomstgenerator' ingevoerd, die de activiteit AANKOMST met bijbehorende tijdsvoorwaarde overneemt van de klasse auto (zie figuur 2.14).

c. Tot dusver hebben we het reële systeem en zijn interactie met de omgeving gemodelleerd. Er resteert nog (de registraties door) de waarnemer die de uiteindelijk gewenste informatie moet aanleveren (zie figuur 2.3). Om de procesbeschrijving zo goed mogelijk te laten aansluiten bij deze figuur 2.3 zullen we voor de waarnemer een aparte componentklasse invoeren. Omdat de waarnemer geen deel uitmaakt van het reële systeem zullen binnen deze componentklasse geen activiteitsklassen voorkomen. De momenten waarop de waarnemer registraties moet verrichten opdat hij de gewenste informatie kan leveren, zullen we in de stroomschema's aangeven met een *. In figuur 2.14 moeten registraties verricht worden van individuele wachttijden, het aantal bediende auto's en het aantal doorgereden auto's opdat de waarnemer de 'gemiddelde wachttijd' en het aantal doorgereden auto's kan bepalen.

2.7 Relatie tussen de drie beschrijvingswijzen

In onderstaand overzicht is de relatie tussen de drie beschrijvingswijzen schematisch weergegeven:

Act.klassen	Ev.beschr.		Procesbeschrijving				Toestandsvariabele
	A	V	generator	auto	rij	pomp	
AANKOMST	X		X				
DOORRIJDEN	X			X			
INRIJ	X			X			RIJLENGTE
UITRIJ	X	X				X	POMPVRIJ en RIJLENGTE
SCHUIFDOOR		X			X		
VERTREK		X				X	POMPVRIJ

Figuur 2.14: De volledige procesbeschrijving van het tankstation

2.8 Het kwantitatieve model

Voor het uitvoeren van simulaties moeten de waarden van de attributen van alle componenten alsmede het aantal componenten van de verschillende permanente componentklassen bekend zijn. Sommige attribuutwaarden kunnen direkt van een waarde worden voorzien. Andere attribuutwaarden zijn slechts in statistische zin bekend, zodat ten behoeve van de simulatie steeds trekkingen uit de bekende verdelingsfuncties moeten worden gedaan. Het kwalitatieve model gecombineerd met deze gegevens noemen we het kwantitatieve model.

2.9 Opgaven

1 Beredeneer de juistheid in de volgorde van activeren door de component 'besturing' in figuur 2.14 □

2 *Over enige tijd zal voor het tankstation van voorbeeld 2.1 een spoorwegovergang komen, welke elk half uur gedurende 5 minuten aan een stuk gesloten zal zijn.
De eigenaresse van het tankstation vraagt zich af wat de invloed hiervan op de gemiddelde wachttijd bij de pomp en het percentage doorrijders zal zijn voor verschillende groottes van de wachtruimte voor de auto's.
Er moet een simulatie uitgevoerd worden met behulp van de eventbeschrijving.
Doorloop hiervoor de volgende stappen:

a. Bepaal de relevante uitgangs-, omgevings-, stuur- en toestandsvariabelen (= stap 2 van werkwijze 7.2).

b. Geef een grafische weergave van het conceptueel model (= stap 3 van werkwijze 7.2).

c.
   - Bepaal per componentklasse de bijbehorende 'activiteiten'.
   - Maak hiervan een doorsnede en som alle activiteitsklassen op.
   - Geef per activiteitsklasse de bijbehorende veranderingen van de toestandsvariabelen.
     (= stap 5 van werkwijze 7.2).

d. Bepaal de eventklassen en geef per eventklasse een verbale eventbeschrijving.

e. Geef, in tabelvorm, voor elke activiteitsklasse aan in welke eventklasse deze voor kan komen.
(= analogon van stap 6 van werkwijze 7.2).

f. Teken het stroomschema van de eventbeschrijvingen.
Ga hierbij uit van figuur 2.8 en detailleer daarbij de eventbeschrijvingen eveneens in de vorm van een stroomschema.
(= analogon van stap 7 van werkwijze 7.2).

g. Geef aan welke metingen waar uitgevoerd moeten worden om aan de gewenste informatie te komen. □

3 In een kroeg staat een toog waaraan plaats is voor maximaal 5 personen. Dorstigen arriveren in de kroeg (met negatief exponentieel verdeelde tussenaankomsttijden).
Dorstigen die bij aankomst geen plaatsje vinden aan de toog lopen door naar de aanpalende kroeg. Als een dorstige bij aankomst ziet dat de tapper niet bezig is met het helpen van een klant, dan waarschuwt de dorstige de tapper dat hij/zij iets wil bestellen. De tapper tapt een drankje, geeft het aan de dorstige waarna deze elders in de kroeg plaatsneemt. Als er nog meer dorstigen aan de toog zitten neemt de tapper de volgende bestelling op, tapt etc. Als er geen dorstigen meer aan de toog zitten, gaat de tapper iets anders doen, totdat een nieuwe dorstige aan de toog plaatsneemt en te kennen geeft iets te willen bestellen.
De bedieningstijd is homogeen verdeeld. De 'rijdiscipline' is FIFO. Men wil een simulatie uitvoeren om te bepalen:

- hoeveel procent van de dorstigen doorloopt,
- de tijd die men gemiddeld moet wachten op een drankje,
- hoeveel procent langer dan een zekere tijd moet wachten.

Maak een procesbeschrijving van deze kroeg aan de hand van de volgende stappen:

a. De stappen a tot en met c van opgave 2.
b. Wijs de activiteitsklasssen eenduidig toe aan de componentklassen (= stap 6 van werkwijze 7.2).
c. Teken het stroomschema van de procesbeschrijvingen van de componentklassen (= stap 7 van werkwijze 7.2).
d. Geef in de stroomschema's aan waar welke grootheden gemeten moeten worden door de 'waarnemer' om de gewenste informatie te kunnen leveren. □

4 Beantwoord opnieuw de vragen a tot en met d van opgave 3 indien de dorstigen zelf hun drankje tappen en na afloop hiervan de langst wachtende aan de toog waarschuwt dat deze aan de beurt is. Vergelijk de resultaten met de resultaten van opgave 3. □

5 Beantwoord weer de vragen a tot en met d van opgave 3 indien er twee tappers zijn die de klanten parallel bedienen. □

6 Beantwoord opnieuw de vragen a tot en met d van opgave 3 indien een dorstige na het verlaten van de toog zijn of haar (van hem?) drankje drinkt en hij of zij daarna, als de dorst nog niet gelest is en de toog nog niet vol is, daar nog één keer plaats neemt om een tweede, tevens laatste, drankje te bestellen.
Indien in het laatste geval de toog wel vol is verlaat de dorstige de kroeg. De duur van het 'drinkproces' is homogeen verdeeld. □

7 In een fabriek komen twee soorten orders binnen volgens een Poisson proces. Alle orders ondergaan dezelfde bewerking op machine A. Orders van soort 1 verlaten daarna de fabriek terwijl orders van soort 2 nog een bewerking ondergaan op machine B. Zolang machine B bezet is, hebben orders van soort 2 voorrang op orders van soort 1 bij machine A. Is machine B onbezet dan hebben orders van soort 1 bij machine A voorrang. De duur van een bewerking van een order op zowel machine A als machine B is homogeen verdeeld.
Bij elke machine is de volgorde waarin de orders van een bepaalde soort bewerkt worden FIFO. Om- en insteltijden van de machines zijn verwaarloosbaar. De leiding van de fabriek wil graag weten hoe de gemiddelde doorlooptijd van elk soort order afhangt van de combinatie van bewerkingstijden op de 2 machines.
Maak van deze fabriek een procesbeschrijving.
Doorloop hierbij de stappen a tot en met d van opgave 3. □

8 *In een bedrijf worden orders met behulp van een machine verwerkt. De orders komen gemiddeld om de 6.5 minuut binnen (de tussenaankomsttijden mogen negatief exponentieel verdeeld verondersteld worden). De bewerkingstijd van een order is homogeen verdeeld tussen 5 en 10 minuten. De volgorde waarin de orders bewerkt worden is FIFO. De machine heeft gemiddeld 10 maal per werkdag van 8 uur (een Poisson proces) te maken met een storing (defect). De storingen treden op tijdens het bewerken van een order of als de machine al geplaagd wordt door een storing. In het laatste geval zal eerst de reeds aanwezige storing verholpen moeten worden en daarna de nieuwe storing. Als een machine bezig is met een order bij het optreden van een storing, kan, nadat de storing verholpen is, met de order verder gegaan worden alsof er geen storing geweest was. Het opheffen van een storing kost 10 tot 20 minuten (homogeen verdeeld).
De bedrijfsleiding vraagt zich af of ze een andere, snellere maar minder betrouwbare, machine zal aanschaffen. De bewerkingstijd van een order zal dan homogeen verdeeld zijn tussen 4 en 8 minuten, terwijl het gemiddeld aantal storingen per werkdag zal toenemen tot 15. Als criterium hierbij zal de gemiddelde doorlooptijd per order gebruikt worden.
Stel in het kader van deze vraagstelling een procesbeschrijving op voor dit bedrijf via de stappen a tot en met d van opgave 3. □

9 In een ijzermijn zijn 4 graafmachines tegelijkertijd in bedrijf. Deze laden rotsblokken met ijzererts erin op vrachtwagens. De vrachtwagens rijden hiermee naar een andere plaats op het terrein waar 4 identieke kraakinstallaties staan opgesteld om het ijzererts uit de rotsblokken vrij te maken.
Nadat een vrachtwagen zijn lading heeft kunnen lossen, rijdt deze via een andere weg terug naar de graafmachines om opnieuw gevuld te worden. De capaciteit van een graafmachine bedraagt 250 ton/uur. De rijtijd van een vrachtwagen van een kraakinstallatie naar de graafmachines is

uniform verdeeld tussen 15 en 25 minuten.
De rijtijd in omgekeerde richting is uniform verdeeld tussen 20 en 40 minuten.
Zowel bij de graafmachines als bij de kraakinstallaties staan bij aankomst de vrachtauto's in een wachtrij opgesteld.
De leiding van de ijzermijn vraagt zich af hoeveel vrachtwagens met een laadvermogen van 25 ton minimaal nodig zijn om de dure graafmachines ieder gedurende een werkdag van 8 uur voor meer dan 90% bezet te houden. Het laden van 1 vrachtwagen kost altijd 6 minuten, het lossen 4.
Geef een procesbeschrijving van deze ijzermijn, volg daarbij de stappen a tot en met d van opgave 3.
N.B. Aan het eind van een werkdag staan alle vrachtwagens in een rij bij de 4 graafmachines. □

Hoofdstuk 3

Simulatievoorbeeld met de hand

3.1 Inleiding

Het doel van dit hoofdstuk is inzicht te krijgen in de wijze waarop een simulatie op implementatienivo verloopt. Dit willen we doen aan de hand van een handsimulatie. Pas in de volgende hoofdstukken komt het implementeren in een programmeertaal aan de orde.

De handsimulatie zal eerst volgens de eventbeschrijving worden uitgevoerd om een duidelijk beeld te geven hoe een en ander in de tijd verloopt. Daarna wordt een uitwerking gegeven volgens een procesbeschrijving.

3.2 Een kwantitatief model van het tankstation

We zullen de handsimulatie baseren op het tankstation van voorbeeld 2.1. Doel van de simulatie is hier de gemiddelde wachttijd van de bediende auto's te bepalen waarbij er van wordt uitgegaan dat er geen doorrijders zijn (maximum rijlengte p wordt op oneindig gesteld). We zullen daarbij gebruik maken van de volgende kwantitatieve gegevens die betrekking hebben op 10 auto's.

autonummer :	1	2	3	4	5	6	7	8	9	10
tussenaankomsttijden :	9	5	1	2	1	2	1	5	1	2
bedieningstijden :	4	5	6	3	2	1	4	2	1	1

De bedieningstijd mag pas bekend worden verondersteld als de auto het systeem binnenkomt.

3.3 De eventbeschrijving

We zullen bij het gebruik van de eventbeschrijving alleen *de gevolgen* van de activiteiten aangeven. Hierbij wordt een onderscheid gemaakt tussen gevolgen voor het reële systeem (= toestandsveranderingen) en gevolgen voor de waarnemer (= registraties door middel van hulpvariabelen).

Bij het tankstation kwamen de volgende eventtypen voor:

- komt een auto het systeem binnen (Aankomst van een auto),
- de pomp komt vrij (Vertrek van een auto).

De gewenste informatie kan worden verkregen door de individuele wachttijd per auto en het aantal bediende auto's te registreren.

De gegevens die we bij de handsimulatie moeten bijhouden zijn dan:

- eventtijdstip
- eventtype (A of V)
- tijdstip volgende Aankomst
- tijdstip volgende Vertrek
- POMPVRIJ
- RIJLENGTE
- NBEDIEND (aantal bediende auto's)
- WACHTTIJDi (wachttijd voor auto i)

Door gebruik te maken van het stroomschema van figuur 2.8 is de simulatie eenvoudig uit te voeren. Zij verloopt als volgt (zie figuur 3.1):

tijdgrootheden				toestandsvar.		hulpvariabelen	
tijd stip	eventtype	tijdst. volg. Aankomst	tijdst. volg. Vertrek	POMP VRIJ	RIJ LENG-TE	NBE-DIEND	WACHT-TIJDi
0	-	9	-	ja	0	0	
9	A auto 1	14	13	neen	0	0	0 (auto 1)
13	V auto 1	14	-	ja	0	1	
14	A auto 2	15	19	neen	0	1	0 (auto 2)
15	A auto 3	17	19	neen	1	1	
17	A auto 4	18	19	neen	2	1	
18	A auto 5	20	19	neen	3	1	
19	V auto 2	20	25	neen	2	2	4 (auto 3)
20	A auto 6	21	25	neen	3	2	
21	A auto 7	26	25	neen	4	2	
25	V auto 3	26	28	neen	3	3	8 (auto 4)
26	A auto 8	27	28	neen	4	3	
27	A auto 9	29	28	neen	5	3	
28	V auto 4	29	30	neen	4	4	10(auto 5)
29	A auto 10	-	30	neen	5	4	

Figuur 3.1: Handsimulatieresultaten volgens de eventbeschrijving (tot en met $t = 29$)

We starten op $t = 0$:
De variabelen krijgen de initiële waarde. Het eerstvolgende aankomsttijdstip is $t = 9$. De 'klok' selecteert het eerstvolgende eventtijdstip (in dit geval $t = 9$).

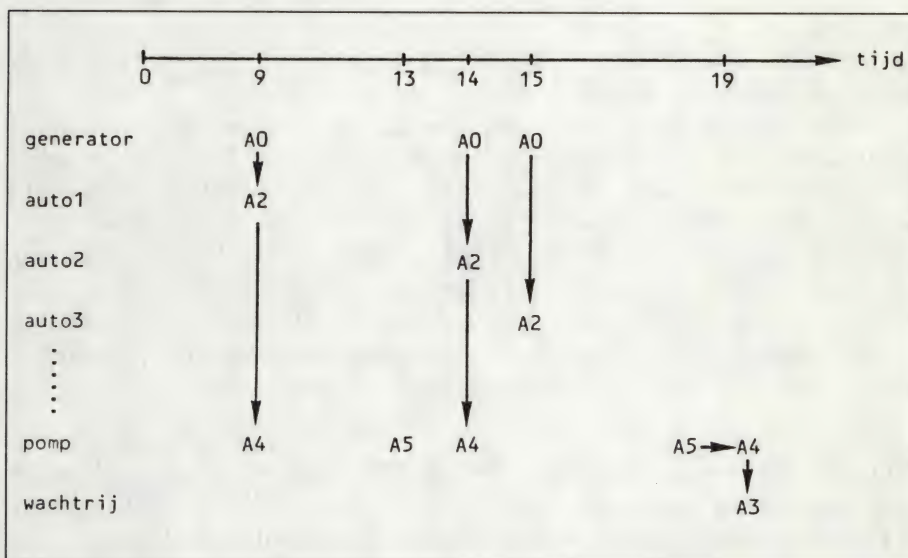
$t = 9$:
Er wordt vastgesteld of het een Aankomst- of Vertrek-event betreft. In dit geval Aankomst; het volgende aankomsttijdstip wordt bepaald, het vetrektijdstip

wordt bepaald en de variabelen worden veranderd overeenkomstig de eventbeschrijving.
Het eerstvolgende eventtijdstip wordt weer door de 'klok' geselecteerd.

$t = 13$:
etc. etc.
Tot slot moet de 'waarnemer' uit de individuele wachttijden en het aantal bediende auto's nog de gemiddelde wachttijd berekenen.

3.4 De procesbeschrijving

Vanwege het quasi-parallel verlopen van de processen van de verschillende componenten wordt de handsimulatie wat ingewikkelder dan bij de eventbeschrijving.
We gaan uit van de procesbeschrijvingen van figuur 2.14.
De samenhang in de tijd van de activiteiten A0 tot en met A5 van de componenten is weergegeven in figuur 3.2.



Figuur 3.2: Samenhang activiteiten van de componenten in de tijd. De pijlen geven de volgorde van de activiteiten op één tijdstip aan.

$t = 0$:
De initialisatie houdt hier in dat op tijdstip $t = 0$ een component 'besturing' wordt geactiveerd. Deze component zal op zijn beurt allereerst (ook op $t = 0$) een waarnemer creëren en activeren waardoor hij zelf passief wordt.

De waarnemer voert geen reële activiteiten uit en zal dus direct weer worden gedeactiveerd. Hierdoor wordt de component 'besturing' weer actief die

dan (nog steeds $t = 0$) een wachtrij creëert en activeert (en daardoor zelf gedeactiveerd wordt).

De wachtrij is initieel leeg (RIJLENGTE = 0) en wordt gedeactiveerd bij 'wacht op UITRIJ'. Via de component 'besturing' wordt vervolgens de pomp gecreëerd en geactiveerd.

De pomp is initieel niet bezet (POMPVRIJ = TRUE) en wordt gedeactiveerd bij 'wacht op INRIJ'. Tenslotte wordt via de component 'besturing' de aankomstgenerator gecreëerd en geactiveerd.

De aankomstgenerator wordt gedeactiveerd op het punt wacht tot TIME = AANKOMSTTIJD (= 9).

De component besturing wordt nu weer actief en wordt vervolgens meteen weer gedeactiveerd.
Nu is er geen enkele component meer actief.
Om de handsimulatie voort te zetten moeten we het eerstvolgende tijdstip weten waarop er weer een activiteit wordt uitgevoerd. Hiertoe lopen we alle opgestarte processen langs en zien dat het eerstvolgende tijdstip (= eventtijdstip) $t = 9$ is.

$t = 9$:
De aankomstgenerator creëert en activeert een auto en wacht tot de volgende aankomsttijd van een auto ($t = 14$).

De gegenereerde auto voert zijn proces uit en neemt plaats in de wachtrij (RIJLENGTE = 1). Tevens wordt de pomp geactiveerd. Deze haalt hem uit de rij (RIJLENGTE = 0), waarbij de wachtrij geactiveerd wordt (nu niet echt nodig omdat RIJLENGTE = 0). Daarna, maar wel op hetzelfde tijdstip $t = 9$, start de pomp de bediening (POMPVRIJ := FALSE) en wacht tot de bediening voorbij is ($t = 13$). Vervolgens, nog steeds op $t = 9$ wordt de auto weer actief die dan met zijn proces 'klaar' is. Voor $t = 9$ zijn nu alle activiteiten uitgevoerd.

Het volgende eventtijdstip wordt geselecteerd (dit behoort bij het proces dat het eerst in de tijd aan de beurt is, in dit geval de pomp).

$t = 13$:
De pomp zorgt ervoor dat de auto vertrekt. Het aantal bediende auto's wordt door de waarnemer verhoogd (NBEDIEND := NBEDIEND + 1). De rijlengte is 0 en de pomp wacht bij 'wacht op INRIJ' waardoor hij passief wordt.

$t = 14$: de aankomstgenerator etc. etc.

De simulatie stopt als er tien auto's zijn bediend. De waarnemer heeft dit aantal bijgehouden. Op grond van de waarde van dit getal zal de pomp op dit moment de component 'besturing' activeren waardoor de simulatie beëindigd wordt.

Dit betekent dat in figuur 2.14 de procesbeschrijving van 'besturing' het driehoekje bevat.

Voor de resultaten zie weer figuur 3.1.

3.5 Opgaven

1 *Voer voor het tankstation een handsimulatie uit volgens de eventbeschrijving voor:

p=3
aankomsttijden : 5, 1, 2, 1, 2, 1, 9, 6, 1, 2
bedieningstijden : 4, 5, 6, 3, 2, 1, 4, 2, 1, 1

□

2 Idem maar nu volgens een activiteitenbeschrijving. □

Hoofdstuk 4

Statistische aspecten van simulatie en modelvalidatie

4.1 Inleiding

Bij de handsimulatie in het vorige hoofdstuk waren zowel de individuele waarden van de kenmerken van de tijdelijke componenten, te weten de tussenaankomsttijden en bedieningstijden, als hun aantal vooraf bekend. In het algemeen zal dit niet het geval zijn. Hoe de waarden voor de kenmerken per individuele component dan kunnen worden verkregen wordt behandeld in 4.2.

Het aantal tijdelijke componenten, of meer algemeen, de duur van een simulatie, hangt direkt samen met de gewenste betrouwbaarheid en nauwkeurigheid van de simulatieresultaten. Hierop wordt nader ingegaan in 4.3.

In 4.4 tenslotte zal kort aandacht worden besteed aan modelvalidatie, de laatste stap voordat met het uitvoeren van de simulatie-experimenten begonnen kan worden. Hiermee is, afgezien van de computerimplementatie, de essentie van de in figuur 1.2 geschetste aanpak behandeld.

4.2 Het genereren van tussenaankomsttijden en bedieningstijden: trekkingen uit verdelingen

Voor het genereren van de voor de simulatie benodigde input zijn er in principe drie mogelijkheden:

a. De in werkelijkheid gemeten waarden van de attributen van de componenten worden opgeslagen in een bestand waar ze later weer uitgelezen worden (zie bijvoorbeeld tabel met getallen voor de 10 auto's in 3.2). We hebben dan te maken met een deterministische input.
b. De waargenomen waarden van de stochastische variabelen kan men opvatten als trekkingen uit een kansverdelingsfunctie. Uit deze kansverdelingsfunctie (bijvoorbeeld gegeven in histogramvorm) worden bij simulatie trekkingen gedaan.
c. De waargenomen waarden van de stochastische variabelen kan men opvatten als trekkingen uit een bekende theoretische kansverdelingsfunctie (bijvoorbeeld negatief exponentiele verdeling, normale verdeling, Erlang verdeling etc.).

Voor een aantal veel voorkomende theoretische verdelingsfuncties bestaan algoritmes voor het doen van trekkingen tijdens een simulatie.

In het onderstaande zullen we laten zien hoe aselecte trekkingen uit verdelingen verkregen kunnen worden door eerst random trekkingen uit een homogene verdeling (0,1) te realiseren en vervolgens deze randomgetallen met behulp van een zogenaamde inverse verdelingsfunctie te transformeren tot een trekking uit de gewenste verdeling.

4.2.1 De pseudo-random generator

Aselecte getallen kunnen op veel manieren verkregen worden, bijvoorbeeld met een zuivere tienkantige dobbeltol. Voor simulatie toepassingen hebben we echter veel aselecte getallen nodig die we bij voorkeur op een later tijdstip weer moeten kunnen reproduceren. Het genereren van zuivere aselecte getallen met behulp van een computer blijkt erg moeilijk te zijn. Wel kunnen bij benadering reeksen aselecte getallen verkregen worden met behulp van algoritmes die eenvoudig op een computer geïmplementeerd kunnen worden. Een dergelijk geïmplementeerd algoritme wordt een pseudo-random generator genoemd.

Het hiervoor aanbevolen basisalgoritme (Shannon 1975, Law and Kelton 1982, Pidd 1984) staat bekend onder de naam lineaire congruente generator.

De reeks integer getallen $\{x_i\}$ wordt hierbij gegenereerd door middel van onderstaande recursieve betrekking:

$$x_{i+1} = (a.x_i + c) \bmod m$$

met: $i = 0, 1, 2 \ldots$
a, c, m zijn constanten
x_0 is de initiële waarde of startwaarde

random getal $r_i = x_i/m$

Getalvoorbeeld: $a = 3, c = 0, m = 5, x_0 = 4$
De getallenreeks wordt dan: $4, 2, 1, 3, 4, 2$, etc.
Voor elke i geldt: $x_i < m$; de reeks is cyclisch, in dit voorbeeld met lengte $m - 1$.
De trekkingen op interval (0,1) zijn:
0,8; 0,4; 0,2; 0,6; 0,8; 0,4 etc.

Er is veel onderzoek gedaan naar geschikte (voor)waarden voor a, c en m.

Er zijn voorwaarden bekend op grond waarvan de getallenreeks 'een volledige periodeduur' heeft, dat wil zeggen elk getal $0, 1, 2, 3, \ldots, m-1$ komt precies één keer voor in de cyclus.

De keuze van het getal m hangt nauw samen met het getalbereik van de computer. Om een zo groot mogelijke periodeduur te halen wordt voor m een priemgetal gebruikt. In de praktijk speelt bij de keuze ook de efficiency van het rekenalgoritme mee (bijvoorbeeld macht van twee rekent 'handig').

Law en Kelton doen bijvoorbeeld de volgende aanbeveling voor een 32-bits machine:

$$m = 2^{31} - 1 = 2.147.483.647; \quad a = 7^5 = 16.807; \quad c = 0.$$

Binnen SIMULA worden de volgende parameters gebruikt:

$$m = 2^{35}; \quad a = 5^{15}; \quad c = 0$$

(merk op dat m hier geen priemgetal is!).

Bij gebruikmaking van een randomgenerator met een grote cycluslengte, waarbij elk integer getal uit de reeks $(1, 2, 3, ..., m - 1)$ één keer in de cyclus voorkomt, kan men in principe een willekeurig getal uit die reeks als startwaarde kiezen. Indien men 'tegelijkertijd' over meerdere random reeksen wil beschikken kan men of met meerdere algoritmen gaan werken of met meerdere reeksen gebaseerd op verschillende startwaarden. Bijvoorbeeld indien het enige verschil tussen twee te vergelijken systemen bestaat uit een machine die per systeem een andere storingsgevoeligheid heeft is het wenselijk dat de randomreeksen van de overige stochastische grootheden ongewijzigd blijven; met andere woorden elke stochastische grootheid moet over een eigen random reeks kunnen beschikken.

Theoretisch bestaat de mogelijkheid dat deze reeksen elkaar overlappen (Kleijnen 1986). Bij een grote cycluslengte is de kans hierop echter uiterst (verwaarloosbaar?) klein. Wil men overlap uitsluiten dan bestaan er algoritmen die niet-overlappendheid waarborgen of men kan gebruik maken van zogenaamde Fishman tabellen waarin geschikte startwaarden verzameld zijn.

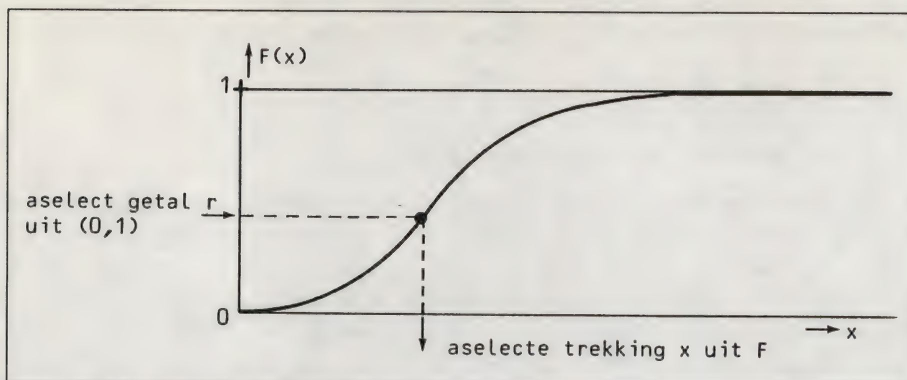
Ook kan men zelf startwaarden verzamelen met behulp van de generator door bijvoorbeeld van een zeer lange reeks getallen om de 10^6 getallen een getal te selecteren en dit te gebruiken als element van een verzameling niet-overlapveroorzakende startwaarden (indien benodigde reeks $< 10^6$).

We komen zo tot het volgende overzicht van eisen/wensen ten aanzien van een pseudo-random generator (Law and Kelton):

a. aselectheid,
b. reproduceerbaarheid,
c. voldoende grote cycluslengte,
d. meerdere random reeksen,
e. efficiency (zowel qua CPU-tijd als geheugenruimte).

4.2.2 Trekkingen uit een kansverdelingsfunctie

Een veelgebruikte methode voor het verkrijgen van aselecte trekkingen uit kansverdelingen zullen we toelichten aan de hand van figuur 4.1.



Figuur 4.1: Een aselecte trekking uit kansverdeling F(x)

De procedure luidt als volgt:

a. Bepaal met behulp van een pseudo-random generator een aselect getal r uit $(0, 1)$,
b. Los x op uit de vergelijking $r = F(x)$ óf $x = F^{-1}(r)$ waarbij F^{-1} de inverse functie van F is.
   De gewenste trekking is dan x.

Voorbeeld 4.1:

Gegeven de negatief exponentiële verdeling met kansdichtheid

$$f(x) = \lambda \,.\, e^{-\lambda x}$$

De kansverdeling is dan: $F(x) = \int_0^t f(t)dt = 1 - e^{-\lambda x}$

We stellen verder: $F(x) = r$

Dus:

$$\ln(1 - r) = -\lambda x$$

$$x = -\frac{1}{\lambda} \,.\, \ln(1 - r)$$

Deze laatste uitdrukking mag statistisch gezien ook geschreven worden als:

$$x = -\frac{1}{\lambda} \,.\, \ln r$$

Het uitvoeren van trekkingen uit andere analytische functies of uit discrete verdelingen of uit continue verdelingen die gespecificeerd zijn door middel van een tabel verlopen in grote lijnen analoog. We merken op dat binnen verschillende talen, zoals SIMULA, reeds standaard procedures aanwezig zijn om aselecte trekkingen uit theoretische verdelingsfuncties te verrichten.

4.3 Hoe lang moeten we simuleren: betrouwbaarheid van simulatieresultaten

In het algemeen zal het gedrag van een te onderzoeken systeem variëren als functie van de tijd. Bijvoorbeeld de wachttijd bij een kassa van een supermarkt zal afhangen van het moment van de dag.

De periode die we simuleren moet overeenstemmen met de voor ons interessante periode van het reële systeem. Het door een simulatierun verkregen gedrag van een systeem als functie van de tijd voor zo'n periode hangt sterk van het toeval af omdat het is gebaseerd op de wisselende duur van de activiteiten van de afzonderlijke componenten. Het gedrag binnen een run kan als het ware worden opgevat als een trekking uit de verzameling mogelijke gedragspatronen van het systeem over de betreffende periode.

Om een zekere betrouwbaarheid van het gedrag als functie van de tijd te verkrijgen moeten we meerdere simulatieruns gaan uitvoeren. Voor elke simulatierun bepalen we het gedrag van het systeem op een gegeven aantal momenten. Vervolgens worden de resultaten van de afzonderlijke runs voor de overeenkomstige momenten gemiddeld.

Met betrekking tot de te simuleren periode zullen we drie gevallen onderscheiden:

a. Simulatie van een systeem dat zich in een zogenaamde stationaire toestand bevindt die eventueel voorafgegaan werd door een opstartfase. In een stationaire toestand zijn de eigenschappen van de relevante grootheden statistisch gezien onafhankelijk van de tijd.
b. Simulatie van de opstartfase van een systeem waarbij de duur van de opstartfase vooraf niet bekend is.
c. Simulatie van een systeem gedurende een van tevoren vaststaande periode. Bijvoorbeeld 1 werkdag van 8 uur.

We zullen nu eerst situatie a verder uitwerken.

Stel met betrekking tot het voorbeeld van het tankstation dat we ons in een stationaire situatie bevinden waarbij we geïnteresseerd zijn in de gemiddelde wachttijd van de auto's.

Stel verder dat we de gemiddelde wachttijd verkregen zouden hebben met behulp van n opeenvolgende wachttijden van auto's uit dezelfde run: $(x_1, x_2, \ldots, x_n)$.

De vraag is nu of deze berekende gemiddelde wachttijd voldoende betrouwbaar is om op grond daarvan beslissingen te nemen.

De serie wachttijden kan worden opgevat als een steekproef, met steekproefgemiddelde $\overline{x}$:

$$\overline{x} = \frac{1}{n} \cdot \sum_{i=1}^{n} x_i$$

en met steekproefvariantie s^2:

$$s^2 = \frac{1}{n-1} \cdot \left(\sum_{i=1}^{n} x_i^2 - \frac{1}{n} \cdot \left(\sum_{i=1}^{n} x_i \right)^2 \right)$$

Om met behulp van deze grootheden een betrouwbaarheidsinterval voor $\mu = E\underline{x}$ te kunnen bepalen is het noodzakelijk dat $\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_n$ onderling onafhankelijk zijn en afkomstig zijn uit dezelfde verdeling (Een stochastische variabele x wordt aangegeven als $\underline{x}$; de operator E staat voor verwachtingswaarde).

Echter in het algemeen, zo ook in het geval van het tankstation zullen $\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_n$ niet onderling onafhankelijk zijn! Een lange wachter zal in het algemeen buren hebben die ook lange wachters zijn.

We kunnen wel een betrouwbaarheidsinterval voor $\mu = E\underline{x}$ berekenen als we over meerdere onafhankelijke trekkingen van $\overline{\underline{x}}$ zouden kunnen beschikken. (Immers ook $E\overline{\underline{x}} = \mu$).

We kunnen hier aankomen door een lange simulatierun te verdelen in een aantal subruns.
Stel de subruns genummerd: $0, 1, 2, \ldots, k$.
Subrun i start met de toestand van het systeem zoals die door subrun $i - 1$ is achtergelaten. De nulde subrun is de zogenaamde aanlooprun waarop verderop nader wordt ingegaan.

Alle subruns (behalve de nulde) worden even lang gekozen. De subrunlengte moet zo groot zijn dat de subrungemiddelden bij benadering onderling onafhankelijk zijn.

De onderlinge afhankelijkheid kan worden uitgedrukt met behulp van de autocorrelatie $\mathrm{COR}(x_i, x_{i+j})$.

Omdat de autocorrelatie het sterkst tot uiting komt in de afhankelijkheid van elkaar opvolgende subrungemiddelden wordt slechts de 1e orde autocorrelatie beschouwd ($j = 1$). Hierop is de Von Neumann ratio $\underline{q}$ gebaseerd (Kleijnen 1986).

$$\underline{q} = \frac{\sum_{n=1}^{n-1} (\overline{\underline{x}}_i - \overline{\underline{x}}_{i+1})^2}{\sum_{n=1}^{n} (\overline{\underline{x}}_i - \overline{\overline{\underline{x}}})^2}$$

n = aantal subruns
$\overline{\underline{x}}_i$ = gemiddelde van subrun i
$\overline{\overline{\underline{x}}}$ = het gemiddelde van de subrungemiddelden

Indien de gemiddelden x van de subruns onderling onafhankelijk zijn en $N(\mu, \sigma^2)$ verdeeld kan worden bewezen dat $\underline{q}$ bij benadering $N\left(2, \frac{4(n-2)}{n^2-1}\right)$ verdeeld is.

We zullen de Von Neumann ratio $\underline{q}$ gebruiken als criterium voor de beoordeling van onderlinge onafhankelijkheid van subruns.

Stel nu dat we op grond van de waarde van $\underline{q}$ de subrungemiddelden $(\underline{\bar{x}}_1, \underline{\bar{x}}_2, \ldots, \underline{\bar{x}}_k)$ onderling onafhankelijk mogen veronderstellen.
Eén subrungemiddelde kan opgevat worden als een trekking uit een onbekende verdeling.
Het gemiddelde van de subrungemiddelden zullen we aanduiden met $\underline{\bar{\bar{x}}}$ en de variantie van de k subrungemiddelden weer met $\underline{s}^2$.
Volgens de centrale limietstelling geldt dat de som van k trekkingen uit een onbekende verdeling (k voldoende groot) bij benadering opgevat mag worden als een trekking uit een normale verdeling.

Zo vinden we met een betrouwbaarheid $(1-\alpha)$, het volgende betrouwbaarheidsinterval voor μ:

$$\underline{\bar{\bar{x}}} - \frac{\underline{s}}{\sqrt{k}} \cdot t_{k-1}\left(\frac{\alpha}{2}\right) < \mu < \underline{\bar{\bar{x}}} + \frac{\underline{s}}{\sqrt{k}} \cdot t_{k-1}\left(\frac{\alpha}{2}\right)$$

waarin:

α = overschrijdingskans
μ = de verwachting van $\underline{x}$
$\underline{\bar{\bar{x}}}$ = het gemiddelde van de subrungemiddelden
$\underline{s}$ = standaardafwijking van de k subrungemiddelden
k = aantal subruns
$t_{k-1}\left(\frac{\alpha}{2}\right)$ = waarde af te lezen uit student's T-verdeling
$\frac{\underline{s}}{\sqrt{k}}$ = standaardafwijking van $\underline{\bar{\bar{x}}}$)

We zien dus dat het aantal subruns van een simulatierun wordt bepaald door de gewenste nauwkeurigheid en betrouwbaarheid van de uiteindelijke uitspraken.
Er bestaan variantiereducerende technieken waardoor men met een kleiner aantal subruns even nauwkeurige uitspraken kan doen. In dit boek wordt daar niet verder op ingegaan (Kleijnen 1986).

Er zal nu worden aangegeven hoe we in principe experimenteel de minimaal benodigde lengte van een aanlooprun kunnen bepalen alsmede de minimale benodigde lengte van de subruns opdat de subrungemiddelden onderling onafhankelijk verondersteld mogen worden. Daarmee wordt tevens het in het begin van deze paragraaf onderscheide punt b behandeld. De betrouwbaarheid van de resultaten van punt c kan eenvoudig worden bepaald door herhaling van runs met de gegeven simulatieduur.

4.3.1 Experimentele bepaling lengte aanlooprun

We zullen in deze paragraaf de aanlooprunproblematiek visueel inzichtelijk maken.

In de praktijk zal men bij de bepaling van de lengte van de aanlooprun veelal werken met de resultaten van proefruns, of met vuistregels gebaseerd op inzichten bijvoorbeeld verkregen uit experimenten van onderstaand type.

De experimenten hebben betrekking op het tankstation nu met een onbeperkte wachtruimte om het aanloopeffect beter zichtbaar te maken. Hierbij is aangenomen dat de tijden tussen de aankomsten bij het tankstation negatief exponentieel verdeeld zijn (met parameter $\lambda = 1/5,5$) en de bedieningstijden constant (5 eenheden).

Voor deze situaties is de bezettingsgraad (dit is de verhouding tussen aankomstintensiteit en maximaal mogelijke bedieningsintensiteit)

$$r = \frac{1/5,5}{1/5,0} = 0,91$$

Een voor de hand liggende aanpak lijkt te zijn om de waarde van de gewenste grootheid (doorlooptijd) voor de opeenvolgende auto's grafisch uit te zetten en te kijken of deze grootheid een 'stationaire toestand' bereikt. Dit is gedaan in figuur 4.2A.

Hierin is geen aanloopeffect zichtbaar. Dit aanloopeffect is misschien wel aanwezig maar wordt dan volledig overheerst door de grote variaties in de individuele doorlooptijden. In de figuren is te zien dat er 'regelmatig' een wachttijd van 0 tijdseenheden optreedt.

(De doorlooptijd is dan gelijk aan de bedieningstijd; volgens de theorie geldt dit gemiddeld voor $1 - r = 0,09$ gedeelte van de totaal beschouwde tijd).

Om de variaties enigszins te onderdrukken en om de tijdas compacter te kunnen weergeven is de aanlooprun verdeeld in aansluitende stukken (zogenaamde subruns) van ieder 100 auto's. Per subrun is de gemiddelde doorlooptijd ($\overline{dlt}$) berekend. De zo verkregen gemiddelde doorlooptijd per subrun is uitgezet in figuur 4.2B. Ook deze resultaten zijn erg fluctuerend, zonder duidelijk zichtbaar aanloopeffect.

Om de variaties nog verder te onderdrukken zijn van 100 verschillende aanloopruns zoals weergegeven in figuur 4.2B (dus ieder bestaande uit 30 subruns van elk 100 auto's) de gemiddelde doorlooptijden van de corresponderende subruns gemiddeld. De zo verkregen resultaten voor het gemiddelde van de gemiddelde doorlooptijden ($\overline{\overline{dlt}}$) staan weergegeven in fig. 4.2 C,D,E voor drie verschillende bezettingsgraden.

We kunnen nu voor $r = 0,91$ en $r = 0,82$ een duidelijk aanloopgedeelte onderkennen. Voor alle drie de gevallen is duidelijk een stationaire fase zichtbaar.

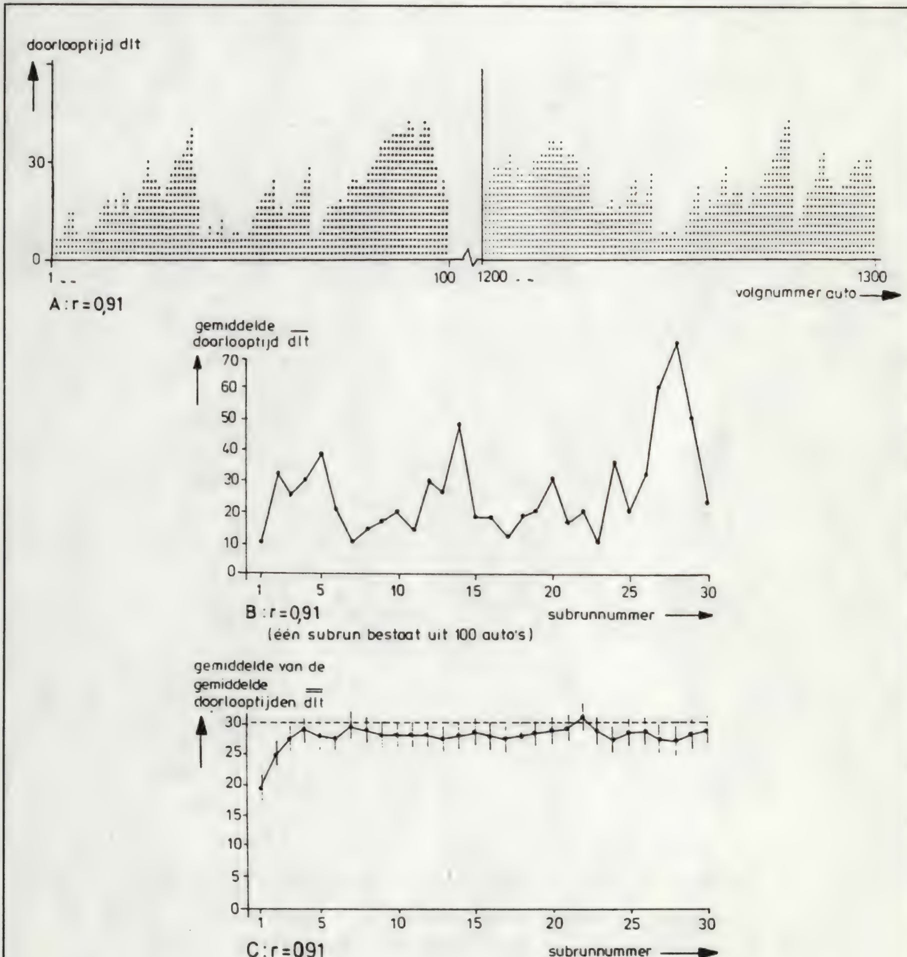
Voor elke r-waarde kan worden afgeleid na hoeveel subruns het subrungemiddelde 'de stationaire waarde' bereikt heeft.

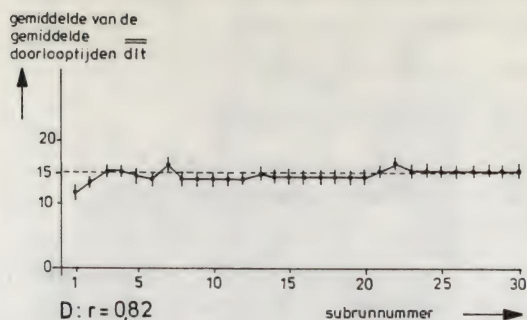
We merken op dat de gevonden 'stationaire waarden' overeenstemmen met de hieronder berekende theoretische waarden.

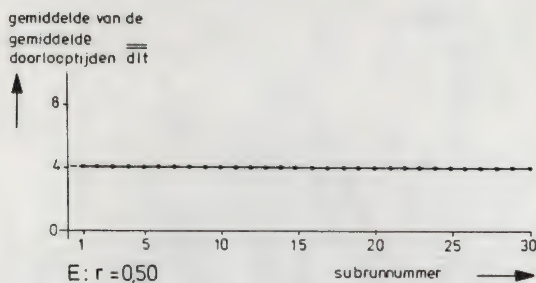
$$
\begin{aligned}
E(\underline{dlt}) &= E(\underline{\text{wachttijd}}) + E(\underline{\text{bedieningstijd}}) \\
&= \left(\frac{r}{2(1-r} + 1\right) \times E(\underline{\text{bedieningstijd}}) \\
&= 30{,}3 \text{ (voor } r = 0{,}91 \text{ en bedieningstijd} = 5) \\
&= 15{,}6 \text{ (voor } r = 0{,}82 \text{ en bedieningstijd} = 4{,}5) \\
&= 4{,}13 \text{ (voor } r = 0{,}5 \text{ en bedieningstijd} = 2{,}75)
\end{aligned}
$$

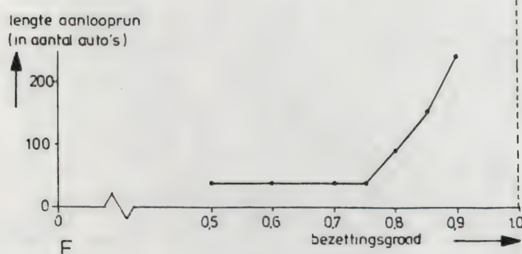
In figuur 4.2C wordt na 300 à 400 auto's de 'stationaire waarde' bereikt. Figuur 4.2F geeft de experimenteel gevonden relatie tussen de lengte van de aanlooprun en de bezettingsgraad (nu echter gebaseerd op een subrunlengte van 25 auto's).

Hieruit blijkt dat de benodigde lengte van de aanlooprun sterk toeneemt bij toenemende bezettingsgraad.









Figuur 4.2: Enkele onderzoekresultaten met betrekking tot de lengte van de aanlooprun. $\overline{dlt}$ = gemiddelde doorlooptijd; r = bezettingsgraad. De stippellijn in figuur C, D en E geeft de verwachtingswaarde aan.

4.3.2 Experimentele bepaling lengte subrun

De problematiek van de bepaling van de noodzakelijke lengte van de subrun voor het geval de simulatie uit een lange run bestaat, wordt hier weer toegelicht aan de hand van het tankstation.

De tussenaankomsttijden worden weer negatief exponentieel verondersteld; de bedieningstijd homogeen tussen 2 en 8 tijdseenheden. De bezettingsgraad r kan worden ingesteld door de keuze van de waarde van de gemiddelde tussenaankomsttijd.

We gaan er vanuit dat er een voldoende lange aanlooprun geweest is zodat we ons in een stationaire situatie bevinden. We willen de betrouwbaarheid van de verkregen waarde voor de gemiddelde doorlooptijd in de stationaire situatie bepalen aan de hand van de gemiddelde doorlooptijden van de verschillende subruns. De subrungemiddelden dienen hiervoor onderling onafhankelijk te zijn. Als criterium hiervoor zal de Von Neumann ratio worden gehanteerd. Bij de uitvoering van het experiment moet nog een keus gemaakt worden voor de subrunlengte en het aantal subruns per run.

In figuur 4.3A zijn voor 10 verschillende subrunlengtes, variërend van 25 tot 1600, de bijbehorende q-waarden (gebaseerd op 100 subruns per run) uitgezet.

Theoretisch geldt dat er sprake is van onderlinge onafhankelijkheid met een betrouwbaarheid van 95% indien:

$$q > 2 - 1,645 \,.\, \sigma_q$$

Omdat het aantal subruns in figuur A gelijk is aan 100 ($n = 100$) luidt het criterium hier

$$q > 1,67$$

Vergelijken we de experimenteel bepaalde q-waarden met dit criterium dan blijkt dat hier subruns met een lengte vanaf 200 onderling onafhankelijk verondersteld mogen worden.

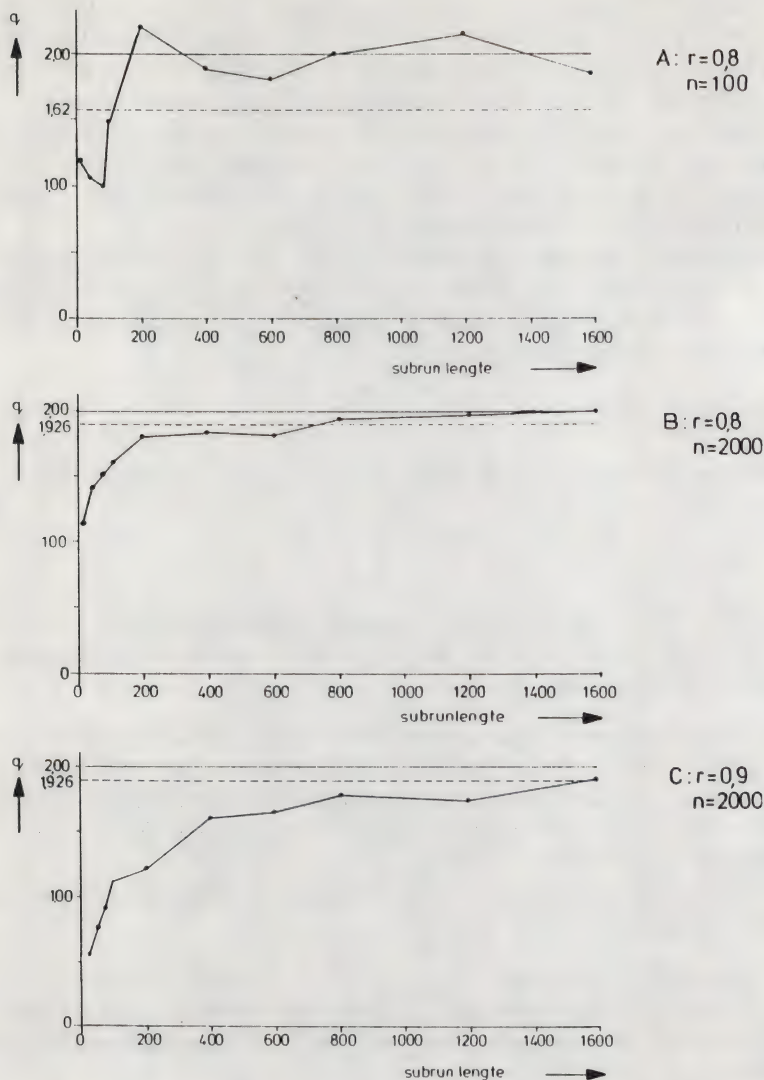
Bij herhaling van dit experiment (met andere random reeksen) blijkt er een vrij grote variatie in het verloop van de curven te zitten. Om toch tot een richtlijn te komen voor de minimaal noodzakelijke subrunlengte is het gewenst dat de variaties meer onderdrukt worden. Dit is gedaan door met een groter aantal subruns te werken ($n = 2000$).

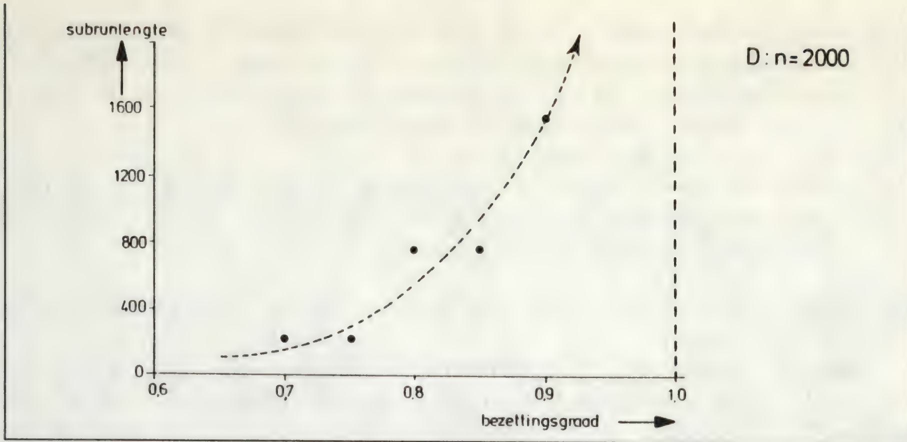
In figuur 4.3B en 4.3C zijn de zo verkregen resultaten voor $r = 0,8$ resp. $r = 0,9$ weergegeven.

In figuur 4.3D zijn een aantal van dergelijke resultaten samengevat. Hieruit blijkt dat de benodigde subrunlengte sterk afhankelijk is van de bezettingsgraad. Voor een bezettingsgraad van ca. 0,85 is de benodigde subrunlengte al ca. 1000 auto's.

De hiervoor beschreven resultaten geven aan hoe men in principe inzicht kan krijgen in de te gebruiken lengtes van de aanlooprun en subruns. Volgens

Law en Kelton (1982) geven resultaten van elementaire één-loketsystemen zoals hier behandeld (zij werken overigens met exponentieel verdeelde tussenaankomsttijden en bedieningstijden) reeds goede indicaties voor de benodigde subrunlengtes bij meer complexe simulatiemodellen.





Figuur 4.3: Enkele onderzoeksresultaten met betrekking tot de subrunlengte ten behoeve van onderling onafhankelijke subruns.
r = de bezettingsgraad; n = aantal subruns; per subrunlengte is een run uitgevoerd bestaande uit n subruns van deze lengte.
De horizontale stippellijnen geven de theoretische ondergrens aan van het 95% betrouwbaarheidsinterval voor q.

Tot slot willen we beklemtonen dat het hiervoor behandelde als 'illustratie' bedoeld is en dat men in praktische situaties minder rigoreus te werk behoeft te gaan. Men zou zich dan kunnen beperken tot het bepalen van de q-factor en nagaan of deze aan het gestelde criterium voldoet, waarbij het aanbeveling verdient het aantal subruns gelijk te kiezen aan 100.

4.4 Modelvalidatie en experimentele resultaten

We zullen de systeemschets van figuur 2.2 gebruiken als leidraad bij de modelvalidatie.

Het is allereerst van belang dat het statistisch gedrag c.q. de waarden van de omgevings- en stuurvariabelen geverifieerd zijn.

Daarnaast is een juiste keuze van de begintoestand, al dan niet verkregen met behulp van een aanlooprun, van belang.

Tenslotte moeten de met het model verkregen waarden van de uitgangsvariabelen overeenstemmen met mogelijk bekende waarden afkomstig uit de praktijk c.q. resultaten uit de theorie. Praktijkgegevens hebben doorgaans betrekking op bestaande of historische situaties, dus modelvalidatie kan dan slechts voor deze situaties plaats vinden.

Ook vergelijking met theoretische resultaten is dikwijls slechts ten dele mogelijk (waarom zouden we anders gaan simuleren?). Dit vergelijken kan op twee manieren:

a. Door het gedrag van het model in extreme situaties te beschouwen. Voor extreme situaties is dikwijls wel een analytische oplossing bekend. Deze oplossing(en) kunnen worden gebruikt als ondergrens/bovengrens voor de beoordeling van de resultaten van het model.
b. Door het model te vereenvoudigen.
Hierdoor kunnen soms situaties worden gecreeerd waarvoor weer analytische oplossingen bekend zijn die gebruikt kunnen worden voor het testen van een deel van het volledige model.

De validatie geschiedt veelal met proefruns die worden opgezet volgens de in 4.3 geschetste aanpak.

Bovendien leveren dergelijke proefruns resultaten die dikwijls gebruikt kunnen worden bij de planning van de serie definitieve simulatie experimenten. Bijvoorbeeld bij de bepaling van de lengte van de aanlooprun en de subruns, en het aantal uit te voeren subruns om een zekere betrouwbaarheid/nauwkeurigheid van de resultaten te bereiken.

Voordat men de definitieve serie experimenten gaat plannen is het gewenst dat men inzicht heeft in de gevoeligheid van het model. Dat wil zeggen dat men een indruk moet hebben in welke mate verandering van de waarden van diverse ingangsvariabelen de waarden van de uitgangsvariabelen beïnvloeden. Ook dit kan men weer onderzoeken met proefruns, waarbij telkens de waarde van slechts een variabele wordt gevarieerd.

Op grond van voorgaande resultaten kan tenslotte een serie (definitieve) experimenten gepland en uitgevoerd worden opdat men de gewenste resultaten met de gewenste nauwkeurigheid tot zijn beschikking krijgt.

Mede op grond van deze resultaten wordt vervolgens besloten hoe men het reëele probleem zal gaan oplossen.

4.5 Opgaven

1 *Waarom zien we in figuur 4.2.E geen duidelijke aanloopverschijnselen? □

2 *Beargumenteer het verloop van de curve in figuur 4.3.D. □

3 *Geef één alternatief voor het gebruik van één lange run indien we geïnteresseerd zijn in een stationaire situatie. □

Hoofdstuk 5

Implementatie van de beschrijvingswijzen in Pascal

5.1 Inleiding

Het doel van dit hoofdstuk is om na te gaan in hoeverre de op verschillende beschrijvingswijzen gebaseerde gekwantificeerde modellen eenvoudig geïmplementeerd kunnen worden in een taal als PASCAL. Dit zullen we doen voor het tankstation uit hoofdstuk 2.

5.2 Implementatie van de activiteitenbeschrijving in Pascal

In 2.4 zijn voor het tankstation de drie activiteitsklassen en het bijbehorende stroomschema weergegeven. Het activiteiten-stroomschema van figuur 2.6 lijkt qua structuur erg op het event-stroomschema van figuur 2.8. In het kader van dit boek verwijzen we daarom wat betreft de implementatieaspecten in PASCAL naar de hierna volgende implementatie van de eventbeschrijving.

5.3 Implementatie van de eventbeschrijving in Pascal

We gaan hierbij uit van de resultaten zoals gepresenteerd in 2.5. Het hieruit afgeleide pseudocode-programma is weergegeven in figuur 5.1.

Bij de verdere uitwerking van het pseudocode-programma zullen een aantal problemen die bij de implementatie van alle drie de beschrijvingswijzen optreden, opgelost moeten worden:

a. Wat betekent 'not klaar' precies, met andere woorden hoe lang moeten we simuleren?
b. Hoe genereren we tussenaankomsttijden en bedieningstijden?
c. Hoe implementeren we een wachtrij?

Op a en b zijn we reeds ingegaan in het vorige hoofdstuk. Met betrekking tot b zullen we aannemen dat uit de analyse van waarnemingen is gebleken dat de tussenaankomsttijden van auto's goed beschreven kunnen worden door

middel van een negatief exponentiële verdeling met een gemiddelde van 4 minuten (met andere woorden het aankomstproces is een Poissonproces met 0.25 gebeurtenissen per minuut). Tevens zullen we aannemen dat de bedieningstijden bij de pomp beschreven mogen worden door middel van een homogene verdeling tussen 1 en 6 minuten.

Met betrekking tot c merken we op dat de in figuur 5.1 genoemde wachtrij dient voor de opvang van de tijdelijke componenten na aankomst. De wachtrij kunnen we implementeren als een array W. De lengte van array W moet minstens gelijk zijn aan het maximale aantal auto's dat op een tijdstip in het systeem kan staan te wachten. Deze lengte moet vooraf bekend zijn! Het uit de wachtrij gaan van een auto houdt in dat de overige aanwezige auto's een plaats 'opschuiven', dus dat $W_i := W_{i+1}, i = 1, 2, \ldots,$ RIJLENGTE.

```
Start + initialisaties;
WHILE not klaar DO
BEGIN selecteer volgende event(klok);
      IF eventklasse=aankomst THEN
      BEGIN genereer volgende aankomsttijdstip;
            IF rijlengte < p THEN
            BEGIN plaats de auto in de wachtrij;
                  IF de pomp vrij is THEN
                  BEGIN haal de auto uit de wachtrij,
                        registreer de wachttijd,
                        schuif overige auto's een plaats door,
                        bezet de pomp,
                        genereer tijdstip vertrek,
                  END;
            END;
            ELSE rijdoor;
      END;
      IF eventklasse=vertrek THEN
      BEGIN verwijder de auto van de pomp,
            maak de pomp vrij,
            IF de wachtrij is niet leeg THEN
            BEGIN haal eerste auto uit de wachtrij,
                  registreer de wachttijd,
                  schuif overige auto's een plaats door,
                  bezet de pomp,
                  genereer tijdstip vertrek;
            END;
      END;
END;
uitvoer gemiddelde wachttijd;
```

Figuur 5.1: Een eventbeschrijving van het tankstation in pseudocode

Het probleem van het vooraf bekend moeten zijn van de maximale rijlengte en het 'doorschuiven' is te voorkomen door gebruik te maken van zogenoemde *lijsten*.

Een lijst kan men voorstellen als een ketting bestaande uit schakels die, met uitzondering van de eerste en laatste schakel, zodanig aan elkaar gekoppeld zijn dat iedere schakel precies één voorganger en één opvolger heeft. In elke schakel wordt bijgehouden wie zijn voorganger en opvolger zijn. Men kan op elke willekeurige plaats in de ketting nieuwe schakels toevoegen èn bestaande schakels verwijderen. Bij het afbeelden van een wachtrij als een ketting komt elke tijdelijke component overeen met een schakel, waarin informatie over die component kan worden opgeslagen.

Veel van de huidige programmeertalen beschikken over mechanismen om dergelijke kettingen op efficiënte wijze te implementeren. Men noemt het dan een *datastructuur* en de acties die men op de datastructuur kan uitvoeren *operaties*.

Alhoewel men in PASCAL zelf lijsten met behulp van het recordtype kan definiëren zullen we dat terwille van de eenvoud in het voorbeeld niet doen.

We zijn nu in staat om het pseudocode programma van figuur 5.1 te detailleren tot een volledig PASCAL programma.

Hieronder volgen de omschrijvingen van de programma-variabelen terwijl in figuur 5.3 het programma wordt gegeven.

variabele	betekenis
. ARRAANK	array met aankomsttijden van auto's
. ARRET	array met eventtijdstippen
. EVTYPE	type van het eerstvolgende eventtijdstip
. GEMTAT	gemiddelde tussenaankomsttijd
. GEMWACHT	somwt/nwacht
. GWS	som gemm. wachttijden
. GWT	gem. wachttijd over de subruns
. GWTKWT	som van de kwadraten van gem. wachttijd over de subruns
. I,J	hulpteller
. INF	een zeer groot getal
. NSUBRUN	subrun teller
. NAANKOMST	aantal aankomsten tot nu toe
. NDOORRIJDEN	teller van het aantal auto's die geen plaats vinden op het emplacement
. NBEDIEND	aantal bediende auto's
. NWACHT	teller van het aantal gemeten wachttijden
. MAXRIJL	maximale lengte wachtrij
. OG,BG	ondergrens/bovengrens van het bedieningstijden interval

. POMPVRIJ	boolean variabele; TRUE als de pomp vrij is, anders FALSE
. RIJLENGTE	actuele lengte van de wachtrij
. SOMWT	som van de wachttijden
. SOMKWT	som van kwadraten van wachttijden
. SUBRUNL	totaal aantal te verwerken auto's per subrun
. TIME	systeemtijd
. VARWACHTGEM	variantie gem. wachttijd over de subruns
. WACHTTIJD	wachttijd van een individuele auto

Figuur 5.2: De in figuur 5.3 gebruikte variabelen met hun betekenis

```
PROGRAM TANKSTATION(INPUT,OUTPUT);
VAR I,NSUBRUN,MAXRIJL,INF,NWACHT,NBEDIEND,SUBRUNL,NDOORRIJDEN,EVTYPE,
    RIJLENGTE : INTEGER;
    TIME,WACHTTIJD,SOMWT,GEMTAT,OG,BG,GWS,GWT,GWTKWT,VARWACHTGEM,
    GEMWACHT : REAL;
    POMPVRIJ : BOOLEAN;
    ARRET : ARRAY[0..1] OF REAL;
    ARRAANK : ARRAY[1..100] OF REAL;

PROCEDURE BEZET;
VAR J : INTEGER;
BEGIN WACHTTIJD:=TIME-ARRAANK[1];
      SOMWT:=SOMWT+WACHTTIJD;
      RIJLENGTE:=RIJLENGTE-1;
      IF RIJLENGTE > 0 THEN
      BEGIN FOR J:=1 TO RIJLENGTE DO ARRAANK[J]:=ARRAANK[J+1] END;
      POMPVRIJ:=FALSE;
      ARRET[1]:=TIME+OG+(BG-OG)*RANDOM;
END;

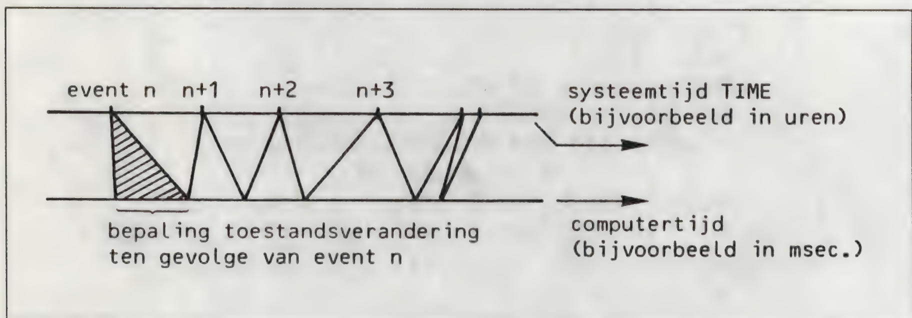
BEGIN (*INITIALISATIES*)
      WRITE('GEEF DE WAARDE VAN MAXRIJL : ');READLN(MAXRIJL);
      WRITELN('GEEF SUBRUNLENGTE,GEMTAT,OG,BG :');READLN(SUBRUNL);
      READLN(GEMTAT);READLN(OG);READLN(BG);
      NSUBRUN:=0;INF:=10000;NWACHT:=0;NBEDIEND:=0;NDOORRIJDEN:=0;
      RIJLENGTE:=0;SOMWT:=0.0;GWS:=0.0;GWT:=0.0;GWTKWT:=0.0;
      VARWACHTGEM:=0.0; GEMWACHT:=0.0;WACHTTIJD:=0.0;
      POMPVRIJ:=TRUE;ARRET[0]:=0.0;ARRET[1]:=INF;
      TIME:=0;ARRET[0]:=TIME-GEMTAT*LN(RANDOM);
      (*BESTURING AANTAL SUBRUNS*)
      WHILE NSUBRUN < 11 DO
      BEGIN (*BESTURING SUBRUNLENGTE*)
            WHILE NBEDIEND < SUBRUNL DO
            BEGIN (*KLOK*)
                  TIME:=INF;
```

```
                FOR I:=0 TO 1 DO IF ARRET[I] < TIME THEN
                BEGIN EVTYPE:=I;TIME:=ARRET[I] END;
                ARRET[EVTYPE]:=INF;
                IF EVTYPE=0 THEN
                BEGIN (*EVENTKLASSE = AANKOMST*)
                      ARRET[0]:=TIME-GEMTAT*LN(RANDOM);
                      IF RIJLENGTE < MAXRIJL THEN
                      BEGIN RIJLENGTE:=RIJLENGTE+1;
                            ARRAANK[RIJLENGTE]:=TIME;
                            IF POMPVRIJ THEN BEZET
                      END
                      ELSE NDOORRIJDEN:=NDOORRIJDEN+1
                END;
                IF EVTYPE=1 THEN
                BEGIN (*EVENTKLASSE = VERTREK*)
                      NBEDIEND:=NBEDIEND+1;
                      POMPVRIJ:=TRUE;
                      IF RIJLENGTE > 0 THEN BEGIN NWACHT:=NWACHT+1;
                                                  BEZET;
                                            END;
                END;
          END;
          (*BEREKENING + UITVOER RESULTATEN*)
          WRITELN('SUBRUNNUMMER = ',NSUBRUN:2);
          IF NWACHT > 0 THEN GEMWACHT:=SOMWT/NWACHT
                        ELSE GEMWACHT:=0;
          WRITELN('GEM.WACHTTIJD SUBRUN = ',GEMWACHT:5:2);
          WRITELN('AANTAL DOORRIJDERS SUBRUN = ',NDOORRIJDEN:4);
          WRITELN;
          IF NSUBRUN > 0 THEN
          BEGIN GWS:=GWS+GEMWACHT;
                GWTKWT:=GWTKWT+GEMWACHT*GEMWACHT;
          END;
          IF NSUBRUN = 10 THEN
          BEGIN GWT:=GWS/10.0;
                WRITELN('GEM.WACHTTIJD TOTAAL = ',GWT);
                VARWACHTGEM:=(GWTKWT-GWS*GWS/NSUBRUN)/(NSUBRUN-1);
                WRITELN('VAR.GEM.WACHTTIJD TOTAAL = ',VARWACHTGEM);
          END;
          NSUBRUN:=NSUBRUN+1;NBEDIEND:=0;SOMWT:=0.0;NWACHT:=0;
          NDOORRIJDEN:=0; GEMWACHT:=0.0;WACHTTIJD:=0.0;
    END;
END.
```

Figuur 5.3: Het Pascal-programma voor het tankstation

Toelichting op het programma:

a. Merk op dat in figuur 5.1 een deel van de eventbeschrijving 'aankomst' identiek is aan een deel van de eventbeschrijving 'vertrek'. Dit identieke gedeelte is geïmplementeerd als de procedure 'BEZET'.
b. De array ARRET bevat de tijdstippen van het eerstvolgende event 'aankomst' (= ARRET[0]) en 'vertrek' (= ARRET[1]). De 'klok' selecteert steeds het eerstkomende event. De systeemtijd TIME wordt dan door de 'klok' gelijkgemaakt aan het tijdstip van het 'eerstkomende event'. Het zal duidelijk zijn dat na het opstarten van het systeem (op TIME = 0) het eerstvolgende event van het type 'aankomst' moet zijn.
c. De systeemtijd TIME geeft de opeenvolgende eventtijdstippen waarop de toestand van het systeem verandert. Tussen de opeenvolgende eventtijdstippen verandert de toestand van het systeem niet; deze tussenliggende tijd kunnen we bij simulatie overslaan. Alhoewel in het systeem de events tijdloos verlopen zal bij simulatie het bepalen van de daarbij optredende toestandsveranderingen wel (computer)tijd kosten. Het onderscheid tussen systeemtijd en computertijd is als volgt te illustreren:



d. RANDOM is een functie in PASCAL voor het genereren van random getallen. (Zie 4.2.1).

Een simulatie bestaat hier uit 11 subruns van elk 500 auto's waarbij de nulde subrun als aanlooprun wordt gebruikt.
Per subrun wordt de gemiddelde wachttijd en het aantal doorrijders berekend. De resultaten van de 11 uitgevoerde subruns staan in de tabel op de volgende pagina.
Uit de gemiddelde wachttijden van de 10 subruns, waarvan wordt aangenomen dat deze onderling onafhankelijk zijn, wordt het gemiddelde en de variantie hiervan berekend.

subrunnr.	gem. wachttijd	aantal doorrijders
0	4,81	51
1	5,54	63
2	5,37	58
4	4,87	43
5	4.90	43
6	5,16	56
7	5,01	45
8	5,12	53
9	5,26	55
0	4,88	38

Het gemiddelde van de gem. wachttijden van run 1 t/m 10: $\overline{\overline{x}} = 5,10$.
De variantie van de gem. wachttijden van run 1 t/m 10 is: $s = 0,35$.
Met $\overline{\overline{x}}$ en s kan nu een betrouwbaarheidsinterval voor $\mu = E\underline{x}$ worden berekend. Volgens de formules van vorig hoofdstuk ligt μ met een betrouwbaarheid van 95% binnen het interval:

$$\overline{\overline{x}} - \frac{s}{\sqrt{10}} . t_9(0,025) < \mu < \overline{\overline{x}} + \frac{s}{\sqrt{10}} . t_9(0,025)$$

$$4,85 < \mu < 5,35$$

5.4 Implementatie van de procesbeschrijving in Pascal

Met betrekking tot procesbeschrijvingen ligt de implementatie in PASCAL moeilijker. Bekijken we het stroomschema van het tankstation van figuur 2.14 dan zien we dat het mogelijk moet zijn de uitvoering van een proces (programma) tijdelijk te onderbreken (zie symbolen 'wacht tot' en 'wacht op'). Eveneens is het wenselijk dat de onderlinge afstemming tussen de verschillende processen (quasi-parallelliteit) eenvoudig geïmplementeerd kan worden. PASCAL beschikt standaard niet over de benodigde faciliteiten om dit direct te implementeren.
Hetzelfde geldt ook voor andere talen zoals FORTRAN, ALGOL, COBOL.

Meer algemeen kunnen we nu de volgende eisen/wensen met betrekking tot een taal voor de implementatie van procesbeschrijvingen formuleren:

a. Het kunnen implementeren van een procesbeschrijving als een module; dit wil zeggen dat van een componentklasse zowel de procesbeschrijving als de data met betrekking tot die componentklasse deel uitmaken van dezelfde module.

b. Het eenvoudig kunnen genereren van de afzonderlijke componenten (elk met eigen attribuutwaarden en individuele procesbeschrijving) uitgaande van de procesbeschrijving en attributen van de componentklasse. (Vergelijk recordoccurence versus recordtype).
c. Het kunnen beschikken over lijstfaciliteiten bijvoorbeeld voor een eenvoudige implementatie van wachtrijen.
d. Het kunnen beschikken over een eenvoudige implementatie van de 'klok', waarbij de eventtijdstippen automatisch door het implementatiehulpmiddel worden afgeleid uit de individuele procesbeschrijvingen.
e. Het kunnen beschikken over een eenvoudige implementatie van 'quasiparallelliteit'.

Een taal die aan bovengenoemde eisen voldoet is de reeds klassieke taal SIMULA die het onderwerp van het volgende hoofdstuk zal vormen.

5.5 Opgaven

1 Verifieer dat het Pascalprogramma uit figuur 5.3 overeenstemt met het pseudocode-programma van figuur 5.1. □

2 *In het Pascalprogramma kan de maximale rijlengte interactief worden opgegeven. Wat is de maximaal toegestane waarde van p? □

3 Relateer de onder 5.4 genoemde eisen aan de specificaties van de procesbeschrijving van het tankstation uit hoofdstuk 2. □

Hoofdstuk 6

Introductie in de taal SIMULA

6.1 Inleiding

Dit hoofdstuk is zodanig opgezet dat het aansluit op de eisen/wensen die aan het eind van het vorige hoofdstuk zijn opgesteld met betrekking tot een taal voor implementatie van een procesbeschrijving. Tevens is getracht dit hoofdstuk zo op te zetten dat het ook als een introductie in de taal SIMULA gebruikt kan worden zonder enige kennis van de voorafgaande vijf hoofdstukken.

De hier gegeven introductie in de taal SIMULA kan interessant zijn voor diegenen die bijvoorbeeld:

- meer achtergrondinformatie willen hebben met betrekking tot informatiesysteem ontwerpmethoden als 'Jackson Structured Development' (Jackson 1983, Cameron 1986),
- achtergrondinformatie willen hebben over bepaalde aspecten van 'Expert Systems' zoals 'frames' (Harmon and King 1985),
- een minder technische inleiding in de taal SIMULA op prijs stellen dan die gegeven is in bijvoorbeeld het standaardwerk van Birtwistle et al. (1973, 1984),
- een (klassiek) voorbeeld willen zien van een object-georiënteerde taal.

Hoe de in 5.4 genoemde wensen ondersteund kunnen worden met beschikbare faciliteiten binnen SIMULA is weergegeven in figuur 6.1.

In 6.2 wordt het class concept geïntroduceerd. Een class kan opgevat worden als een recordtype gecombineerd met een stuk programma dat daarop operaties kan uitvoeren. Het class concept sluit aan bij de in hoofdstuk 2 behandelde procesbeschrijving van een componentklasse.

In 6.3 wordt het begrip subclass of gespecialiseerde class behandeld. Dit is met name van belang voor een goed begrip van de structuur van een aantal door SIMULA beschikbaar gestelde faciliteiten.

In 6.4. wordt de implementatie van wachtrijen in de vorm van lijsten beschreven. Tevens wordt behandeld hoe het quasi-parallel verlopen van processen ondersteund wordt door de subclass PROCESS. Ook wordt hier kort ingegaan op de implementatie van het klokmechanisme.

Vervolgens worden in 6.5 enkele taalkundige aspecten van SIMULA behandeld, terwijl in 6.6 op de binnen SIMULA beschikbare statistische functies wordt ingegaan.

Tot slot volgt in 6.7 een SIMULA-implementatie van het tankstation (van voorbeeld 2.1).

Implementatie gewenst van:	beschikbare faciliteiten binnen SIMULA:
- data + operaties in één module (attributen + procesbeschrijving)	- (sub)class concept
- wachtrij ('lijst')	- gespecialiseerde class (LINKAGE CLASS HEAD) (LINKAGE CLASS LINK)
- quasi-parallelliteit	- gespecialiseerde class (LINK CLASS PROCESS)
- klokmechanisme	- gespecialiseerde class (zgn. eventketting opgebouwd met behulp van LINKAGE CLASS HEAD LINKAGE CLASS LINK)

Figuur 6.1: Een overzicht van het gebruik van beschikbare faciliteiten binnen SIMULA ten behoeve van het implementeren van procesbeschrijvingen

Een aantal voorbeelden en figuren in dit hoofdstuk zijn gebaseerd op het werk van Birtwistle et al. (1984) en van Papazoglou et al. (1984).

6.2 Het class concept

Een (object)class kan men zich voorstellen als een recordtype gecombineerd met een stuk programma (de zogenaamde classbody) eventueel inclusief procedure(s). Alle variabelen (d.w.z. parameters en locale variabelen) die bij creatie van een object beschikbaar komen, worden in een record van het betreffende type opgeslagen. Dit record is bereikbaar met behulp van een pointer, in SIMULA een *reference variabele* geheten.

Een laatste overeenkomst met PASCAL is, dat er meerdere objecten (ook wel genoemd 'occurences', 'voorkomens', 'exemplaren') van eenzelfde objectclass kunnen worden gecreëerd, zoals meerdere records van eenzelfde recordtype in PASCAL.

Bij creatie van een nieuw object in SIMULA wordt de classbody meteen uitgevoerd. Nadat alle statements binnen de body uitgevoerd zijn (later zal blijken dat ook onderbrekingen mogelijk zijn) kan deze niet meer opnieuw worden doorlopen. Het record met variabelen blijft echter bestaan zolang er een pointer naar verwijst.

We beginnen in voorbeeld 6.1 met een CLASS declaratie genaamd 'voertuig'. Dit is een type definitie, er 'gebeurt dus nog niet echt wat'.

Voorbeeld 6.1: *declaratie class voertuig*

```
CLASS voertuig(bouwjaar, gewicht); INTEGER bouwjaar; REAL gewicht;
BEGIN REAL belasting, gemsnelheid;
      INTEGER kmstand;
      REF(voertuig) voorganger;

      IF bouwjaar < 1970 OR gewicht < 1000.0
      THEN belasting := 500.0
      ELSE belasting := 0.5*gewicht;
      gemsnelheid := 0.0; kmstand:=0;
END**voertuig**;
```

De hier gedeclareerde class bevat geen procedures. De class voertuig kent een 6-tal attributen of variabelen: 2 parameters (bouwjaar en gewicht) en een 4-tal locale variabelen (belasting, gemsnelheid, kmstand en voorganger), die gezamenlijk de recordomschrijving vormen. Bouwjaar en gewicht worden parameters genoemd omdat bij creatie van een object van deze klasse hiervoor waarden moeten worden opgegeven. 'Voorganger' is een REFerence variabele (pointer) die naar een ander object, hier toevallig van dezelfde class voertuig, kan wijzen. In de classbody wordt de belasting voor een gecreëerd voertuig berekend. In de terminologie van hoofdstuk 2 komt de class-body overeen met de procesbeschrijving en het class-record met de attributen van een componentklasse. De actuele waarden van de variabelen bepalen de actuele toestand van een individueel object van die class. De body (of activiteitenlijst) beschrijft de activiteiten die een object zelf uitvoert.

Van de class voertuig kan nu een object gecreëerd en geactiveerd worden door middel van de opdracht NEW.

Voorbeeld 6.2: *creatie en activatie van een object van de class voertuig*

```
NEW voertuig(1921, 1800.0);
```

bouwjaar	1921	
gewicht	1800.0	
belasting	500.0	} class-record
gemsnelheid	0.0	
kmstand	0	
voorganger	NONE	

```
IF bouwjaar < 1970 OR gewicht < 1000.0
THEN belasting := 500.0
ELSE belasting := 0.5 * gewicht;
gemsnelheid := 0.0; kmstand := 0;
```

} class-body

In bovenstaande figuur zijn de waarden van de variabelen van het gecreëerde object weergegeven evenals diens 'class-body'. Het is voor de gebruiker net alsof elk gecreëerd object over een eigen class-body beschikt (in werkelijkheid wordt

dit maar één keer opgeslagen; het SIMULA-systeem zorgt ervoor dat vanuit elk gecreëerd object de bijbehorende class-body toegankelijk is).

Na de instructie NEW wordt hier meteen begonnen met de uitvoering van de classbody waarbij de actuele parameterwaarden worden ingevuld. Nadat alle statements van de body zijn uitgevoerd kan men de waarden van de variabelen, opgeslagen in het record, alleen nog bereiken en veranderen via een pointer die naar het betreffende object verwijst. Voor iedere pointer geldt dat deze slechts naar één type object kan verwijzen. Dit object-type moet bij de declaratie van de pointer vermeld worden.
Bijvoorbeeld:

```
REF(voertuig) eend, volvo, mijnauto;
```

waarbij eend, volvo, mijnauto pointers zijn die kunnen verwijzen naar objecten van de class voertuig.

De initiële waarde van de pointers is NONE, dit wil zeggen er is nog niet aangegeven naar welke objecten van de class voertuig door hen verwezen wordt. Dit geven we symbolisch als volgt weer:

eend	NONE

volvo	NONE

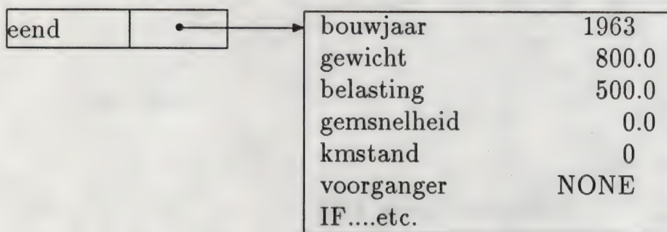
mijnauto	NONE

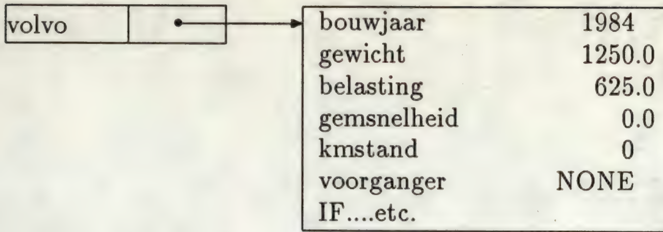
De voertuigen waarnaar met de naam eend respectievelijk volvo verwezen wordt, worden bijvoorbeeld gecreëerd en geactiveerd met:

```
eend   :- NEW voertuig(1963, 800.0);
volvo  :- NEW voertuig(1984, 1250.0);
```

Het teken :– staat voor toekenning van een object aan een referentievariabele, zoals := staat voor toekenning van een waarde aan een variabele.

Zo ontstaat de volgende situatie:

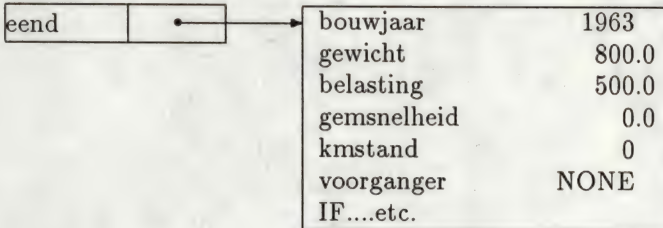




Stel dat nu het volgende statement wordt uitgevoerd:

```
mijnauto :- volvo;
```

Dan verandert de situatie als volgt:



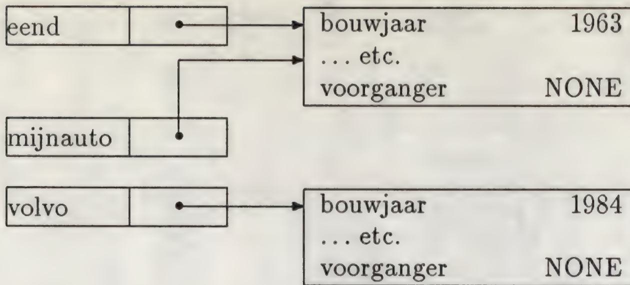


Via een referentievariabele kan de waarde van een attribuut van een object worden verkregen door middel van de punt-notatie (net als bij records in PASCAL). Hiervan is gebruik gemaakt in:

```
IF mijnauto.belasting > 600.0 THEN mijnauto :- eend;
```

Het is niet altijd nodig om voor elk object een afzonderlijke referentievariabele te declareren. Door in een class een referentievariabele op te nemen, zoals voorganger in de class voertuig, kan men zelf eenvoudig een ketting van voertuigen maken. Als we nu over de gegevens met betrekking tot al onze vroegere auto's willen kunnen blijven beschikken, dan kan dat door bij aankoop van een nieuwe auto als volgt te handelen.

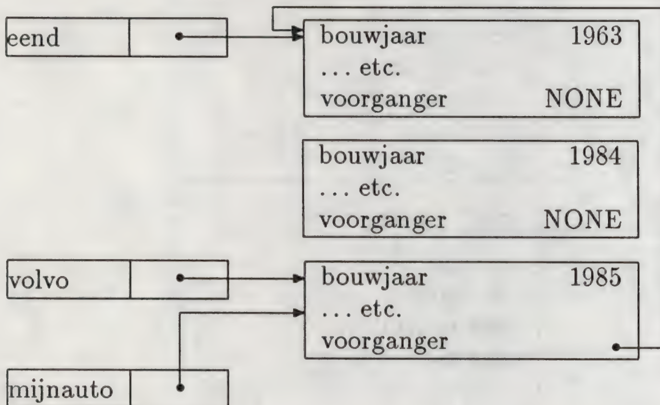
Stel de oude situatie was de volgende:



en de verandering is:

```
volvo :- NEW voertuig(1985, 1250.0);
volvo.voorganger :- mijnauto;
mijnauto :- volvo;
```

dan wordt de nieuwe situatie:



We zien dat het object dat in de oude situatie met volvo werd aangeduid niet meer met behulp van een pointer bereikbaar is, en dus nutteloos geworden is. Binnen SIMULA worden niet bereikbare objecten automatisch verwijderd uit het systeem (zogenaamde 'garbage collection'). In de nu volgende voorbeelden zullen we de schematische tekeningen van objecten achterwege laten.

Referentievariabelen kunnen onderling worden vergeleken met het symbool == (beide wijzen naar hetzelfde object) of =/= (beide wijzen niet naar hetzelfde object). Voorbeeld:

```
IF mijnauto == eend THEN OUTTEXT ("het kon erger");
IF mijnauto =/= volvo THEN OUTTEXT ("jammer");
```

SIMULA kent ook de negatie NOT zodat de laatste opdracht ook geschreven kan worden als:

```
IF NOT(mijnauto == volvo) THEN OUTTEXT ("jammer");
```

De parameters en locale variabelen van objecten kunnen worden vergeleken met behulp van het gebruikelijke = teken. Bijvoorbeeld:

```
IF eend.belasting = volvo.belasting THEN OUTTEXT ("oneerlijk");
```

De toestand van een object kan in principe ook van buitenaf veranderd worden. Bijvoorbeeld door het volgende statement:

```
volvo.bouwjaar := 1700;
```

Dit geeft aanleiding tot een ongewenste toestand aangezien er in 1700 nog geen volvo gebouwd was.

Om dit soort ongewenste toestandsveranderingen te voorkomen en om er voor te zorgen dat de procesbeschrijving en dus de programma's waaruit deze is opgebouwd overzichtelijk blijven, verdient het aanbeveling alle veranderingen van waarden van attributen van een object van buitenaf, plaats te laten vinden d.m.v. binnen dat object gedeclareerde functies of procedures.

In deze functies/procedures legt men dan niet alleen de wijze vast hoe de attributen gewijzigd moeten worden maar ook de voorwaarden waaronder evenals het toegestane bereik van de attribuutwaarden (constraints). Het louter inspecteren van de waarden van attributen kan men zonder enig gevaar wel met behulp van de puntnotatie uitvoeren aangezien hierdoor de toestand van het bijbehorende object niet (ongewenst) kan veranderen.

Stel dat voor een voertuig de variabelen kmstand en gemsnelheid aan het eind van elke rit moeten worden aangepast. Dan kunnen we de class voertuig uitbreiden met een procedure 'verplaats' die voor deze aanpassingen zorgt. Als bijvoorbeeld ook vaak de leeftijd van de auto gebruikt wordt, kan voor het berekenen hiervan een functie worden gedeclareerd.

In voorbeeld 6.3 is het voertuig van voorbeeld 6.1 uitgebreid met de procedure verplaats en de functie leeftijd.

Merk op dat binnen een functie een waarde wordt toegekend aan een variabele met dezelfde naam als de functie.

Hierbij moet het type van de functie overeenkomen met het type van deze variabele.

Merk op dat een procedure of functie ook als een attribuut van een object wordt beschouwd, zodat deze net als een variabele met de puntnotatie kan worden benaderd.

Mijnauto kan nu eenvoudig verplaatst worden d.m.v. de procedureaanroep:

```
mijnauto.verplaats(100.0, 2.0);
```

Hierdoor wordt de kmstand met 100 verhoogd en de gemiddelde snelheid wordt 50. De ouderdom van mijnauto in 1985 kan eenvoudig bepaald worden via

```
ouderdom := mijnauto.leeftijd(1985);
```

Voorbeeld 6.3: *declaratie class voertuig met procedure en functie*

```
CLASS voertuig(bouwjaar, gewicht); INTEGER bouwjaar; REAL gewicht;
BEGIN REAL belasting, gemsnelheid;
      INTEGER kmstand;
      REF(voertuig) voorganger;

      PROCEDURE verplaats(afstand, tijd); REAL afstand, tijd;
      BEGIN kmstand:=kmstand+afstand;
            gemsnelheid:=afstand/tijd;
      END;

      INTEGER PROCEDURE leeftijd(jaar); INTEGER jaar;
      BEGIN IF jaar > bouwjaar
            THEN leeftijd:=jaar-bouwjaar
            ELSE leeftijd:=0;
      END;

      IF bouwjaar < 1970 OR gewicht < 1000.0
      THEN belasting:=500.0
      ELSE belasting:=0.5*gewicht;
      gemsnelheid:=0.0; kmstand:=0;
END**voertuig**;
```

6.3 Specialisatie: subclasses

Stel nu dat er niet alleen sprake is van voertuigen, zoals beschreven in voorbeeld 6.3, maar dat er ook bijzondere voertuigen voorkomen bijvoorbeeld vrachtwagens. Voor een vrachtwagen zijn niet alleen alle eigenschappen van een voertuig van toepassing maar bovendien de volgende 'attributen':

- (de parameter) assental,
- (de variabele) lading,
- (de procedures) laad en los,
- (de functie) asdruk.

Men zegt wel dat vrachtwagen een specialisatie is van voertuig of omgekeerd dat voertuig een generalisatie is van vrachtwagens en niet-vrachtwagens.

Het komt meer voor dat bijzondere begrippen gevormd worden uit meer algemene. Men spreekt dan van specialisatie-hiërarchieën. In SIMULA is het mogelijk specialisatie-hiërarchieën te vormen door de naam van de algemene 'class' als prefix te plaatsen voor de declaratie van de subclass. Zo kan de subclass vrachtwagen eenvoudig worden gevormd uit de class voertuig zoals weergegeven in voorbeeld 6.4.

Voorbeeld 6.4: *declaratie class vrachtwagen door middel van prefixing met behulp van de class voertuig*

```
voertuig CLASS vrachtwagen(assental); INTEGER assental;
   BEGIN REAL lading;

         PROCEDURE laad(vracht); REAL vracht;
         BEGIN lading:=lading+vracht END;

         PROCEDURE los(vracht); REAL vracht;
         BEGIN IF vracht < lading
               THEN lading:=lading-vracht
               ELSE lading:=0.0;
         END;

         REAL PROCEDURE asdruk;
         BEGIN asdruk:=(gewicht+lading)/assental END;

         lading:=0.0
   END**vrachtwagen**;
```

Hierna volgen enige declaraties en opdrachten waarbij de subclass vrachtwagen voorkomt:

```
REF(vrachtwagen) daf;
daf :- NEW vrachtwagen(1975, 8000.0, 3);
daf.verplaats(500.0, 8.0);
daf.laad(2000.0);
IF daf.asdruk > 2000.0 THEN waarschuwing;
```

daf | • ⟶

bouwjaar	1975
gewicht	8000.0
belasting	4000.0
gemsnelheid	62.5
kmstand	500
voorganger	NONE
body en procedures voertuig algemeen	
assental	3
lading	2000.0
body en procedures specifiek voor vrachtwagen	

Meer algemeen: Stel dat class B een subclass is van class A.
Declaratie:

```
   CLASS A(FPA).....;
A  CLASS B(FPB).....;
```

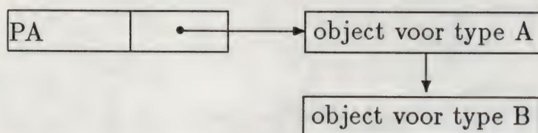
FPA en FPB zijn de formele parameters van class A respectievelijk class B. Creatie van een nieuw object van type class A hebben we in het voorgaande steeds voorgesteld als:

```
PA :- NEW A(waarden FPA);
```



Binnen SIMULA wordt bij creatie van een nieuw object van het type class B automatisch ook een nieuw object van type class A gecreëerd: deze zijn onlosmakelijk met elkaar verbonden. Bij de aanroep moet men daarom zowel de parameterwaarden behorende bij class A als die behorende bij class B aangeven, en wel als volgt:

```
PB :- NEW B(waarden FPA, waarden FPB);
```



Zoals bij een niet-prefixed class meteen na aanmaak van een object de body doorlopen wordt, zo zal dit ook bij de prefixed class gebeuren en wel in de volgorde class A - class B, dus beginnend bij de hiërarchisch hoogste.

Voor de parameters, de locale variabelen en de functies/procedures, (dus de attributen) van class A en class B geldt dat deze als het ware in een groot record worden opgenomen: de combinatie van de twee afzonderlijke records A en B.

De door SIMULA beschikbaar gestelde hulpmiddelen zijn gestructureerd volgens het specialisatie concept. Deze hulpmiddelen zijn opgenomen in een soort bibliotheek (class SIMULATION). Hierover kan worden beschikt door een simulatieprogramma te 'prefixen' met het woord SIMULATION (zie voorbeeld 6.12).

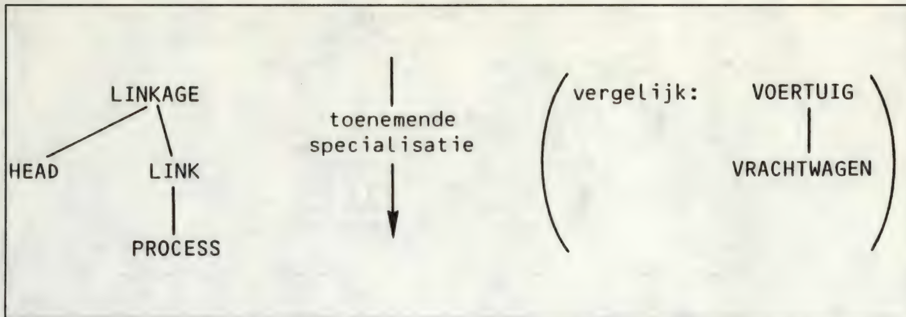
6.4 Systemclass SIMULATION

De belangrijkste binnen de systemclass SIMULATION beschikbare classes zijn LINKAGE, LINK, HEAD en PROCESS.

De eerste drie kunnen gebruikt worden om lijsten te implementeren, de vierde bij implementatie van procesbeschrijvingen.

1. De class LINKAGE biedt een basisstruktuur om circulaire lijsten te implementeren.
2. De class LINK bevat faciliteiten om elementen van een circulaire lijst op te kunnen vatten als componenten van een wachtrij.
3. De class HEAD maakt het mogelijk om één element van een circulaire lijst te beschouwen als de 'beheerder' van de wachtrij.
4. Binnen de class PROCESS zijn faciliteiten gedefinieerd om objecten quasi-parallel hun body's te kunnen laten uitvoeren.

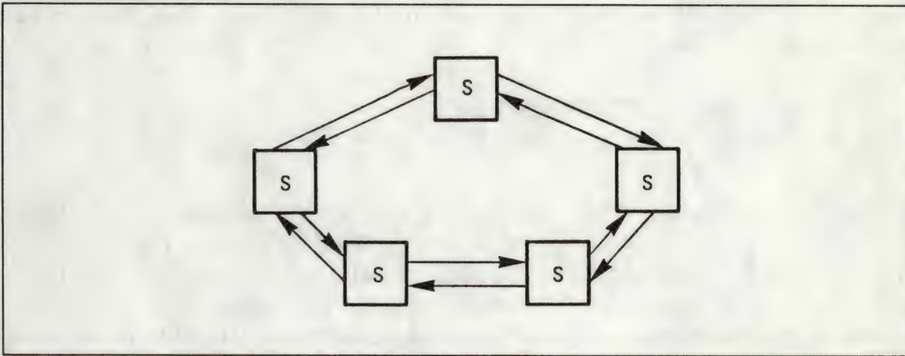
De specialisatie-hiërarchie van de genoemde classes is in figuur 6.2 weergegeven.



Figuur 6.2: Een schets van de hiërarchische structuur van de systemclass SIMULATION

6.4.1 Classes LINKAGE, LINK en HEAD

Een geschikt hulpmiddel voor het implementeren van onder andere wachtrijen is de dubbele, circulaire lijst (two way circular list). Omdat in zo'n lijst elk element een verwijzing heeft naar diens voorganger en opvolger kan men deze lijst in twee richtingen doorlopen zonder dat men een begin of een eind tegenkomt (Figuur 6.3).

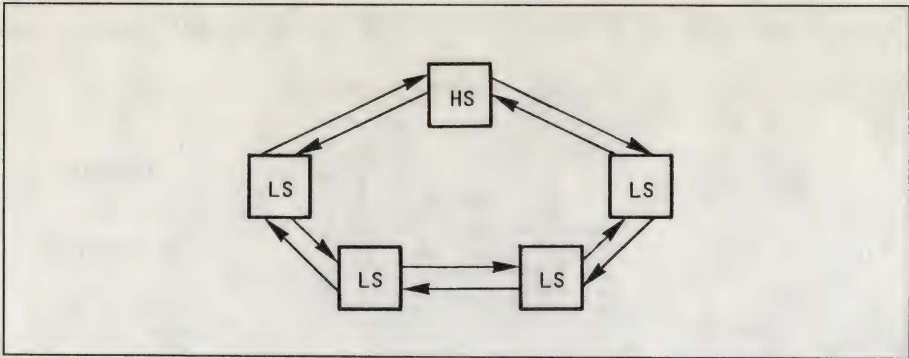


Figuur 6.3: Een dubbele circulaire lijst bestaande uit vijf elementen S

Een dergelijke implementatie van een lijst wordt ook wel een ketting genoemd, opgebouwd uit schakels.

De class LINKAGE biedt de faciliteiten om de ketting in twee richtingen te doorlopen.

Om een lijst als implementatie van bijvoorbeeld een wachtrij te kunnen gebruiken moet er een begin en een eind aan de lijst onderkend kunnen worden. Daarom doorbreken we de ketting door één schakel te specialiseren tot beheerschakel (HEAD) en de overige schakels tot zogenaamde linkschakels (LINK), zie figuur 6.4, waarin HS de beheerschakel en LS de linkschakels weergeven.



Figuur 6.4: Een dubbele circulaire lijst bestaande uit de beheerschakel HS en vier linkschakels LS

In SIMULA zijn de class LINKAGE en de subclasses LINKAGE CLASS HEAD en LINKAGE CLASS LINK standaard beschikbaar.

In figuur 6.5 is een dubbele circulaire lijst meer gedetailleerd weergegeven.

De ketting bestaat uit een HEAD met verwijzers naar de eerste en de laatste schakel van de ketting, respectievelijk suc (successor) en pred (predecessor) genaamd, en een verzameling linkschakels, ook met verwijzers suc en pred.

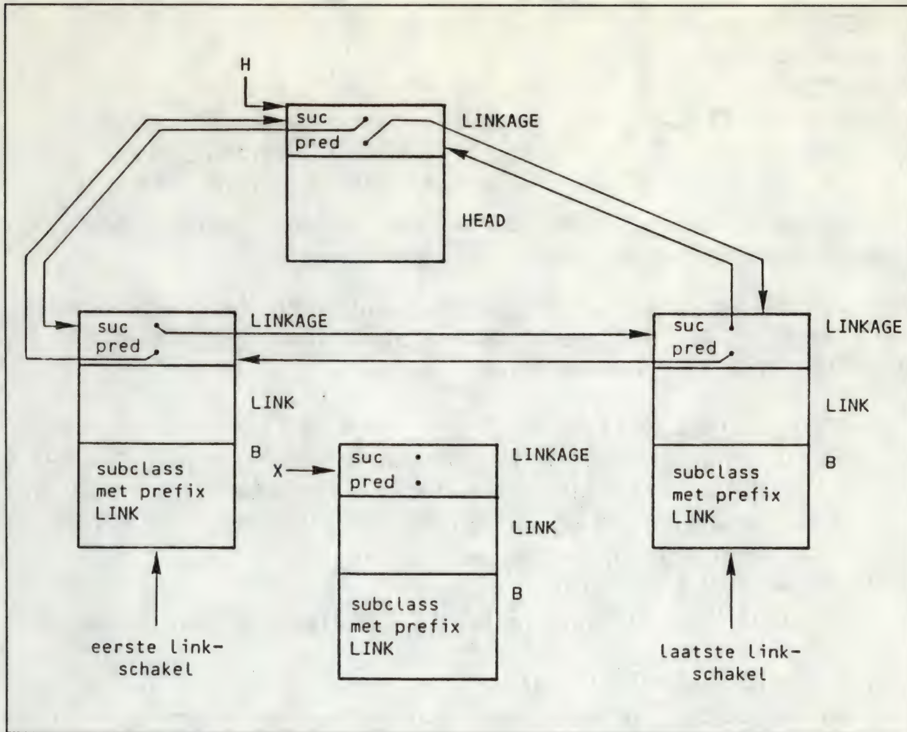
Via de onderstaande binnen de class LINKAGE gedefinieerde procedures kan men een ketting 'doorlopen':

1. REF(LINK) PROCEDURE **SUC**;
2. REF(LINK) PROCEDURE **PRED**;

Deze twee procedures leveren een pointer naar de opvolger respectievelijk de voorganger van een bepaalde schakel in de ketting en zijn beschikbaar via de puntnotatie. Bijvoorbeeld L2 :- L1.SUC, waarbij L1, L2 van het type LINK zijn, heeft als resultaat dat L2 verwijst naar de opvolger van L1. Indien het resultaat geen linkschakel oplevert, dus niet van het type REF(LINK) is, levert dit statement voor L2 NONE op. (Dit vanwege REF(LINK) voor de PROCEDURE SUC EN PRED).

De class HEAD bevat een vijftal procedures die betrekking hebben op de bijbehorende ketting:

1. REF(LINK) PROCEDURE **FIRST**;
   levert een verwijzing naar de eerste linkschakel in de ketting, mits de ketting niet leeg is, anders de waarde NONE,
2. REF(LINK) PROCEDURE **LAST**;
   levert een verwijzing naar de laatste linkschakel in de ketting, mits de ketting niet leeg is, anders de waarde NONE,



Figuur 6.5: Voorbeeld van een kettingstruktuur binnen SIMULA bestaande uit een object van de class HEAD en twee objecten van de class LINK. Het derde object van de class LINK maakt geen deel uit van de ketting. M.b.v. pointer X is deze bereikbaar.
Voor het bovenstaande zijn de volgende declaraties nodig:
REF(HEAD) H;
REF(B) X;
LINK CLASS B.
De subclass B bevat het class-record en de class-body zoals gedefinieerd door de gebruiker.

3. INTEGER PROCEDURE **CARDINAL**;
   levert het aantal linkschakels in de ketting,
4. BOOLEAN PROCEDURE **EMPTY**;
   levert de waarde TRUE als de ketting leeg is (dit wil zeggen geen linkschakels bevat) en anders de waarde FALSE,
5. PROCEDURE **CLEAR**;
   verwijdert alle linkschakels uit de ketting.

De procedures kunnen weer met behulp van de puntnotatie benaderd worden. Voorbeeld:

```
REF(HEAD) H;
H :- NEW HEAD;
H.CLEAR;
OUTINT(H.CARDINAL, 4)
```

(commando voor het uitprinten van H.CARDINAL, een integer getal, waarvoor 4 posities beschikbaar zijn)

De class LINK zorgt ervoor dat de class die van de prefix LINK wordt voorzien de eigenschappen van een linkschakel krijgt.

Voor het manipuleren met linkschakels bevat LINK een viertal procedures (hieronder wordt uitgegaan van de declaraties REF(LINK) L1, L2; REF(HEAD) H; REF(LINKAGE)x):

1. PROCEDURE **OUT**:
   L1.OUT verwijdert object L1 (dat van (een subclass van) class LINK moet zijn) uit de ketting waarvan het deel uitmaakt. Indien L1 zich niet in een ketting bevindt gebeurt er niets. Een LINK-object kan zich slechts in één ketting tegelijk bevinden.
2. PROCEDURE **INTO**(H);
   L1.INTO(H) roept eerst L1.OUT aan en voegt vervolgens L1 als laatste linkschakel toe in ketting H.
3. PROCEDURE **PRECEDE**(x);
   L1.PRECEDE(L2) roept eerst L1.OUT aan en voegt vervolgens L1 toe aan de ketting waarvan L2 deel uitmaakt en wel als voorganger van L2. Als L2 niet in een ketting zit is het effect hetzelfde als L1.OUT. Merk op dat L1.PRECEDE(H) hetzelfde effect heeft als L1.INTO(H).
4. PROCEDURE **FOLLOW**(x);
   Analoog aan procedure PRECEDE met als verschil dat door L1.FOLLOW(L2) L1 de opvolger van L2 wordt.

Met behulp van de systemclasses HEAD en LINK kunnen in het geval van het tankstation van voorbeeld 2.1 (zie figuur 2.14) de auto's en de wachtrij eenvoudig worden beschreven, zoals weergegeven in voorbeeld 6.5 waarbij nog geen rekening is gehouden met de interactie tussen de componentklasse 'auto' en 'pomp'.

(TIME in voorbeeld 6.5 is een standaard functieprocedure, zie 6.4.2, en geeft op elk moment de systeemtijd aan)

De aankomst van een auto wordt gegenereerd door de opdracht:

```
NEW auto;
```

Hierdoor wordt een object van de CLASS auto gecreëerd en geactiveerd. Dit object initialiseert zelf zijn aankomsttijd, berekent zijn bedieningstijd en neemt, mits er ruimte is op het emplacement, plaats in de wachtrij.

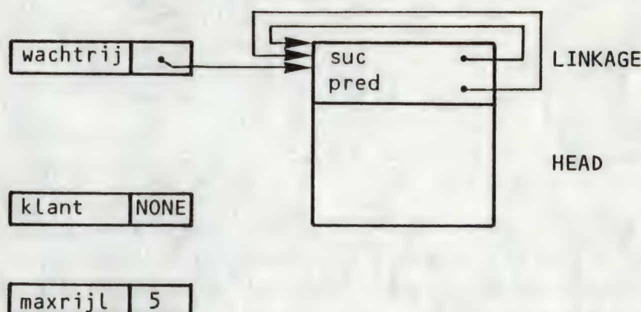
Voorbeeld 6.5: *De procesbeschrijving van 'auto' en 'wachtrij' in SIMULA*

```
LINK CLASS auto;
BEGIN REAL aankomsttijd, bedieningstijd;

      REAL PROCEDURE wachttijd;
      BEGIN wachttijd:= TIME-aankomsttijd
      END;

      aankomsttijd:= TIME;
      bedieningstijd:= trekking uit homogene verdeling;
      IF wachtrij.CARDINAL < maxrijl THEN INTO(wachtrij);
END**auto**;

REF(HEAD) wachtrij;
REF(auto) klant;
INTEGER maxrijl;
maxrijl:= 5;
wachtrij :- NEW HEAD;
```



Het uit de rij nemen van de eerste auto in de figuur op de volgende pagina wordt beschreven door:

```
IF NOT (wachtrij.EMPTY) THEN
BEGIN klant :- wachtrij.FIRST;  (zie stippellijn in figuur)
      klant.OUT;
END;
```

6.4.2 Class PROCESS

De tot nu toe behandelde (sub)classes hebben alle de eigenschap dat de body van objecten hiervan meteen na creatie van het object wordt uitgevoerd (geactiveerd). Dit gebeurt in een ononderbroken executie van de statements van de body. Men kan zo dus niet twee objecten tegelijkertijd (parallel) 'actief' laten zijn. Dit is echter wel noodzakelijk voor bijvoorbeeld procesbeschrijvingen waarin synchronisaties met de systeemtijd ('wachten tot of gedurende') of met andere procesbeschrijvingen ('wachten op') voorkomen.

De class PROCESS maakt het nu mogelijk objecten quasi-parallel actief te laten zijn. Dat wil zeggen men kan de uitvoering van de body (activiteitenlijst) van een object onderbreken en op een later tijdstip hervatten. Meer algemeen, men kan de afhandeling van de activiteiten van de verschillende objecten naar wens indelen (schedulen). Hiertoe moet men de class waartoe dit object behoort voorzien van de prefix PROCESS.

Omdat de class PROCESS een subclass is van de class LINK (zie figuur 6.2), hebben objecten van de class PROCESS tevens alle eigenschappen van de class LINK.

De class PROCESS bevat een variabele EVTIME (eventtime), waardoor aan elk PROCESS object een (event)tijdstip toegekend kan worden. Voor alle objecten met een waarde voor EVTIME wordt door het SIMULA systeem een speciale ketting onderhouden waarin de objecten naar opklimmende waarde van EVTIME gesorteerd zijn. Deze ketting is de zogenaamde *event*ketting en elk object daarin is een eventnotitie.

Bij elke eventnotitie hoort precies één PROCESS-object dat op het betreffende moment geactiveerd moet worden. Het SIMULA systeem zorgt voor de koppeling hiertussen door middel van een pointer van een object uit de eventketting naar het bijbehorende PROCESS-object (zie figuur 6.6).

Iedere haak in de ketting stelt een bepaald eventtijdstip voor. Alle PROCESS-objecten die niet passief of terminated (zie hierna) zijn, zijn in de ketting opgehangen aan een van de in chronologische volgorde gesorteerde haken.

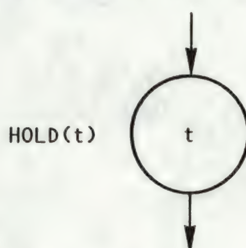
Het tijdstip behorende bij een haak bepaalt het tijdstip waarop het object weer actief zal worden.

De linkerhaak behoort bij de systeemtijd TIME en hieraan hangt het actieve proces. Is dit proces afgehandeld (terminated of suspended) en bevat deze haak geen andere PROCESS-objecten meer, dan wordt deze haak verwijderd en de eerstvolgende haak is aan de beurt (de systeemtijd springt naar het volgende eventtijdstip).

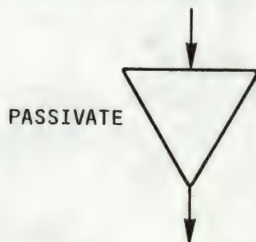
Het eerste object in de eventketting wordt het CURRENT object genoemd. De status van dit object wordt aangeduid met *actief*, omdat de uitvoering van de opdrachten van zijn activiteitenlijst nu wordt voortgezet.

Het current object kan zijn actieve status op drie manieren verliezen:

1. HOLD(t)
   De waarde van EVTIME van het object wordt met t verhoogd. Daardoor krijgt het object een andere plaats in de eventketting. Zijn status luidt nu *opgeschort* (suspended, zie object A in fig. 6.7B).
   HOLD(t) kan worden opgevat als de implementatie van het onderstaande specificatie symbool uit figuur 2.12 (wachten *tot* of *gedurende*):

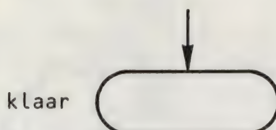


2. Door een PASSIVATE opdracht wordt een object uit de eventketting genomen. EVTIME van dit object heeft nu geen waarde meer, de status van het object is *passief* (zie fig. 6.7C).
   PASSIVATE kan worden opgevat als de implementatie van het onderstaande specificatie symbool uit figuur 2.12 (wachten *op*):



3. Nadat alle statements van de body zijn uitgevoerd krijgt het object (evenals alle niet-PROCESS objecten) de status *klaar* (terminated).

   In de stroomschema's gesymboliseerd door (zie figuur 2.12):



Het SIMULA systeem ruimt van tijd tot tijd alle objecten op, die klaar zijn en waarnaar geen enkele referentie meer bestaat (garbage collection). Dit betekent dat het door deze objecten gebruikte geheugen weer vrij komt.

Een opgeschort object wordt vanzelf weer actief als het de voorste in de eventrij geworden is. Een passief object E, waarnaar een pointer verwijst, kan weer actief worden doordat een ander object een ACTIVATE E opdracht uitvoert. Daardoor wordt object E current (helemaal vooraan in de eventketting geplaatst) en krijgt EVTIME van object E dezelfde waarde als EVTIME van het vorige current object (zie fig. 6.7D).
In stroomschema's wordt ACTIVATE gesymboliseerd door (zie figuur 2.12):

ACTIVATE — — — — →

Als een PROCESS object gecreëerd wordt, *krijgt het de status passief*, zodat een expliciete ACTIVATE opdracht nodig is voordat met uitvoering van de body begonnen wordt.

Raakt de eventketting leeg dan resulteert dat in een runtime error, de simulatie stopt.

In onderstaand overzicht zijn alle standaard procedures die specifiek betrekking hebben op PROCESS objecten opgenomen (ze zijn gedefinieerd in de class SIMULATION):

1. REF(PROCESS) PROCEDURE **CURRENT**;
   levert een verwijzing naar het voorste object in de eventketting.
2. REAL PROCEDURE **TIME**;
   levert de actuele waarde van de systeemtijd.
3. PROCEDURE **HOLD**(t); REAL t;
   maakt dat het current object opgeschort wordt tot tijdstip TIME + t (t > 0). Als dat tijdstip is aangebroken wordt de volgende opdracht uit de activiteitenlijst van het object uitgevoerd.
4. PROCEDURE **PASSIVATE**;
   verwijdert het current object uit de eventketting (waardoor het passief wordt) en gaat verder met het nieuwe current object. Als de eventketting leeg wordt ontstaat een runtime error.

5. PROCEDURE **WAIT**(s); REF(HEAD) s;
   het current object wordt achterin ketting s geplaatst en voert een PASSIVATE opdracht uit. Ketting s is niet de eventketting.
6. PROCEDURE **CANCEL**(x); REF(PROCESS) x;
   PROCESS object x wordt uit de eventketting genomen en wordt passief. Het effect is overeenkomstig PASSIVATE, met dien verstande dat PASSIVATE alleen door het betreffende object zelf kan worden uitgevoerd.

De beschikbare (RE)ACTIVATE statements zijn:

7a. **ACTIVATE** x;
   x wordt het nieuwe current object. De systeemtijd blijft onveranderd. De opdracht heeft alleen effect als x passief is.
7b. **REACTIVATE** x;
   x wordt current, ook als x actief of opgeschort is.
7c. **[RE]ACTIVATE** x AT t [PRIOR]
   x wordt in de eventketting geplaatst met EVTIME = t. Als er een opgeschort object is met EVTIME = t dan wordt x daarachter geplaatst, tenzij de optie PRIOR is vermeld.
7d. **[RE]ACTIVATE** x DELAY t [PRIOR]
   equivalent met AT TIME + t.
7e. **[RE]ACTIVATE** x BEFORE y
   Als y een actief of opgeschort object is dan wordt x voor y in de eventketting geplaatst met dezelfde EVTIME als y.
7f. **[RE]ACTIVATE** x AFTER y
   als 7e, maar nu achter y.

Ook deze procedures en statements kunnen aan de hand van een figuur zoals 6.7 geïllustreerd worden.

Voor de procedures HOLD, PASSIVATE en WAIT geldt dat ze betrekking hebben op het object waarin ze zijn opgenomen. Indien ze gebruikt worden m.b.v. de puntnotatie, dan geldt dat

objectnaam.procedurenaam

betrekking heeft op het object dat met de betreffende objectnaam geïdentificeerd wordt. Voor de instructies ACTIVATE, REACTIVATE, CANCEL en de varianten hierop geldt dat deze betrekking hebben op het aangegeven object.

Een object kan alleen geïdentificeerd worden via een pointer die naar dit object verwijst. Om objecten naar zichzelf te laten verwijzen kent SIMULA de opdracht THIS.
Bijvoorbeeld:

ACTIVATE x AFTER THIS classnaam

activeert object x meteen nadat het object waarin de betreffende ACTIVATE is opgenomen passief geworden is. Voor de volledigheid:

ACTIVATE x AFTER current

heeft hetzelfde effect.

De pomp(bediende) van het tankstation in voorbeeld 2.1 kan nu als volgt worden gedeclareerd als een PROCESS-class in SIMULA:

Voorbeeld 6.6: *De procesbeschrijving van 'pompbediende' in SIMULA*

```
PROCESS CLASS pompbediende;
BEGIN REF(auto) klant;
     WHILE TRUE DO
     BEGIN IF wachtrij.EMPTY THEN PASSIVATE;
           klant :- wachtrij.FIRST;
           klant.OUT;
           HOLD(klant.bedieningstijd);
     END;
END;
```

In het besturingsprogramma komen de volgende declaraties en statements voor:

```
REF(HEAD) wachtrij;
REF(pompbediende) bediende;
wachtrij :- NEW HEAD;
bediende :- NEW pompbediende;
```

Toelichting:
Na de creatie van de bediende, als object van PROCESS CLASS pompbediende, zal deze beginnen met de status passief aan te nemen. Na activering zal hij telkens de eerste auto uit de wachtrij nemen en verdere acties uitstellen tot na de bedieningstijd van de betreffende klant. Als hij merkt dat de wachtrij leeg is, wordt hij passief. Om hem uit die eventuele passiviteit te wekken zal een auto die als eerste in de rij plaats neemt, de opdracht 'ACTIVATE bediende' moeten geven (in voorbeeld 6.5. is hierin nog niet voorzien, maar in voorbeeld 6.12, aan het einde van dit hoofdstuk wel).

Binnen SIMULA is ook het besturingsprogramma (Zie procesbeschrijving BESTURING van figuur 2.14) een PROCESS-object (onder de naam MAIN). Hierdoor is het mogelijk de duur van de simulatie vast te leggen in het besturingsprogramma.

Dit kan op twee manieren:

a. De totale tijdsduur die men wil simuleren is vooraf bekend. Het besturingsprogramma MAIN kan dan worden opgeschort met behulp van HOLD(simulatieduur), zie voorbeeld 6.12.
b. De totale tijdsduur die men wil simuleren is niet vooraf bekend maar hangt af van een of andere voorwaarde (bijvoorbeeld 2000 orders ver-

werkt in een job-shop). Dan moet het besturingsprogramma MAIN passief worden gemaakt (opschorten voor onbepaalde tijd) met behulp van een PASSIVATE in het besturingsprogramma. Op het moment dat aan de betreffende voorwaarde voldaan is moet vanuit het object waar dat wordt geconstateerd het besturingsprogramma worden geactiveerd door middel van ACTIVATE MAIN.

6.5 Taalkundige aspecten van SIMULA

In deze paragraaf zal een overzicht van enkele belangrijke taalelementen gegeven worden. Achtereenvolgens zullen behandeld worden: sleutelwoorden, declaraties, expressies, statements en de scope van variabelen.

6.5.1 Sleutelwoorden

Binnen SIMULA mogen de hierna opgesomde woorden niet als variabele-, functie-, procedure- of classnaam gebruikt worden.

ACTIVATE	ELSE	IS	REAL
AFTER	END	LABEL	REF
AND	EQV	NAME	STEP
ARRAY	FALSE	NEW	SWITCH
AT	FOR	NONE	TEXT
BEFORE	GOTO	NOTEXT	THEN
BEGIN	IF	OR	THIS
BOOLEAN	IMP	OTHERWISE	TRUE
CHARACTER	IN	PRIOR	UNTIL
CLASS	INNER	PROCEDURE	VALUE
COMMENT	INSPECT	QUA	VIRTUAL
DELAY	INTEGER	REACTIVATE	WHEN
DO			WHILE
NOT			

6.5.2 Declaraties

Alle variabelen, functies, procedures en classes, die in een programma worden gebruikt, moeten van te voren worden gedeclareerd. De volgende typen declaraties zijn van belang:

- INTEGER : Variabelen van dit type kunnen alleen gehele getallen als waarde bevatten. Na declaratie hebben ze initieel de waarde 0.
  Declaratie:
  INTEGER var1, var2, ..., varn;

- REAL : Variabelen van dit type kunnen tot een bepaalde nauwkeurigheid reële getallen bevatten. Na declaratie hebben deze initieel de waarde 0.0.
  Declaratie:
  REAL var1, var2, ..., varn;

- BOOLEAN : Boolean variabelen kunnen als waarde TRUE of FALSE hebben. Na declaratie hebben Boolean variabelen initieel de waarde FALSE.
  Declaratie:
  BOOLEAN var1, var2, ..., varj;

- REFERENCE : Reference variabelen (pointers) kunnen als waarde NONE of een verwijzing naar een object bevatten. NONE betekent dat de betreffende pointer geen object aanwijst. Het object dat door een pointer wordt aangewezen moet van het zelfde type zijn als opgegeven bij de declaratie. Initieel hebben pointers na declaratie de waarde NONE.
  Declaratie:
  REF(classnaam) pointer1, pointer2, ..., pointeru;

- CHARACTER : Charactervariabelen kunnen als waarde een willekeurig symbool (letter, cijfer etc.) bevatten. Na declaratie bevatten ze initieel een onbekende waarde.
  Declaratie:
  CHARACTER var1, var2, ..., varm;

- ARRAY : Van de bovenstaande typen variabelen kunnen array's gevormd worden. De declaratie van één-dimensionale array's is als volgt: (Voor meer-dimensionale arrays zie de ALGOL literatuur)
  Declaratie:
  Type ARRAY arr1, arr2, ..., arrn(AE1:AE2);

  waarbij type staat voor INTEGER, REAL, BOOLEAN, REF(classnaam) of CHARACTER.
  AE1 en AE2 zijn aritmetische expressies van het type INTEGER die de beneden- en bovengrens voor de array index aangeven. De betreffende expressies moeten op het moment van declaratie een toegestane waarde opleveren: AE2 > = AE1.

- PROCEDURES EN FUNCTIES:
  Declaratie van een functie:
  Type PROCEDURE procedurenaam(parameterlijst); declaratie parameterlijst; BS;

  of

Type PROCEDURE procedurenaam; BS;

waarbij BS staat voor Blockstatement (zie 6.5.4). Indien 'type' wordt weggelaten spreken we van een procedure en kan de procedurenaam in het aanroepende programma als statement worden opgenomen.
Wordt bijvoorbeeld 'type' niet weggelaten dan levert de aanroep van de functieprocedure een waarde op welke toegekend wordt aan de procedurenaam en spreken we van een functie. Type geeft aan van welk type de door de functie berekende en aan de functienaam toegekende waarde is.
Type moet net als bij array's van het soort INTEGER, REAL, BOOLEAN, REF(classnaam) of CHARACTER zijn. Wordt in de functiebody BS geen waarde toegekend aan de functienaam dan levert de procedurenaam bij aanroep de voor het betreffende 'type' initiële waarde op. Een functienaam moet bij aanroep in het programma altijd rechts van het assignment-teken (:= of :–) in een expressie voorkomen.

- CLASS : De declaratie van een class lijkt sterk op die van een procedure.
Declaratie:
Prefixclassnaam CLASS classnaam (parameterlijst); declaratie parameterlijst; BS;

of

Prefixclassnaam CLASS classnaam; BS;

Indien 'prefixclassnaam' wordt ingevuld dan is de class 'classnaam' een subclass. Wordt er geen 'prefixclassnaam' opgegeven dan is 'classnaam' een 'zelfstandige' class. De 'prefixclassnaam' moet de naam van een andere class zijn die ofwel in hetzelfde programmablok of in een programmablok dat dit omvat gedeclareerd is, ofwel gedeclareerd is in de prefix van het gehele programma. Bijvoorbeeld 'PROCESS' mag als 'prefixclassnaam' gebruikt worden als het programma 'geprefixed' is met 'SIMULATION' aangezien 'PROCESS' als class gedeclareerd is in de systeemclass 'SIMULATION'.

De bovenstaande typedeclaraties mogen elkaar in willekeurige volgorde opvolgen. Men hoeft dus bijvoorbeeld niet eerst alle integer variabelen te declareren, daarna alle real variabelen etc.

6.5.3 Expressies

Binnen SIMULA bestaan vier soorten expressies: aritmetische (rekenkundige), character-, object- (pointer) en boolean expressies; en twee soorten toekenningstekens :- voor objectexpressies en := voor alle andere expressies. Het geheel aan expressies in SIMULA is veel uitgebreider dan hieronder behandeld. Een volledig overzicht valt buiten de opzet van dit boek. Zie hiervoor Birtwistle.

- **Aritmetische expressies (AE):**

  Aritmetische expressies kan men opbouwen m.b.v. variabelen, arrayelementen en functies van het type INTEGER of REAL en de volgende operaties:

+	optelling
−	aftrekking of negatie
/	deling (real)
*	vermenigvuldiging
//	gehele deling (restterm vervalt)
**	machtverheffen

  Het toekenningsteken is := .
  Binnen SIMULA zijn standaard onder andere de volgende functies beschikbaar. (Dit betreft reëelwaardige functies, uitgezonderd de laatste twee):

ABS(x)	absolute waarden van x
ARCTAN(x)	arctangens van x
COS(x)	cosinus van x
SIN(x)	sinus van x
TAN(x)	tangens van x
EXP(x)	e-macht van x
LN(x)	natuurlijke logaritme van x
SQRT(x)	vierkantswortel van x
SIGN(x)	teken van x: −1,0 of 1
ENTier(x)	dichtsbijzijnde integerwaarde kleiner of gelijk aan x

  Bij het aanroepen van functies moet men hun eventueel gedeclareerde parameters van een waarde van het juiste type voorzien.
  Bijvoorbeeld:

  hoogte := 2 * COS(hoek/2).

  Waarbij hoogte en hoek variabelen van het type REAL zijn.

- **Characterexpressies (CE)**

  Characterexpressies leveren als waarde een karakter.
  Het toekenningsteken voor een characterexpressie is := waarbij het karakter tussen quotes staat.

Bijvoorbeeld:

Klasse := 'N'.

Alle functies die CHARACTER als 'type' hebben zijn in character expressies toelaatbaar.

- **Objectexpressies (OE)**

Een object expressie levert als resultaat een object op. Mogelijke objectexpressies zijn:

NONE	geen object toegekend
NEW classnaam;	indien geen parameters nodig
NEW classnaam(parameterwaarden);	
	indien parameters aanwezig. Deze parameterwaarden kunnen ook weer via expressies worden verkregen.
THIS classnaam;	levert verwijzing naar current object op.
p	pointer naar een object

Het toekenningsteken is :- .

Bijvoorbeeld:

p :- NEW A   met A een classnaam

Rechts van het :- teken zijn alle gedefinieerde functies toegelaten die als type REF(classnaam) hebben, evenals alle functies die in pararagraaf 6.4 voor LINK en PROCESS objecten zijn gedefinieerd. Men moet erop letten dat het objecttype (classnaam) dat een objectexpressie oplevert van dezelfde soort is als de variabele waaraan de waarde wordt toegekend. Uitzondering hierop is 'NONE' dat voor alle objecttypen voldoet.

- **Boolean expressie (BE)**

Boolean expressies leveren als waarde TRUE of FALSE op en kunnen m.b.v. de boolean operatoren uit onderstaand overzicht geconstrueerd worden.

Overzicht verschillende boolean operatoren:

arit. expr:	AE1 < AE2	kleiner
	AE1 > AE2	groter
	AE1 <= AE2	kleiner of gelijk
	AE1 >= AE2	groter of gelijk
	AE1 = AE2	gelijk
	AE1 <> AE2	ongelijk
object expr:	OE1 == OE2	verwijzen naar zelfde object
	OE1 =/= OE2	verwijzen niet naar zelfde object
char. expr:	CE1 = CE2	karakters gelijk
bool. expr:	BE1 AND BE2	en
	BE1 OR BE2	of
	NOT BE1	negatie
	BE1 EQU BE2	gelijk
	TRUE	altijd waar
	FALSE	altijd niet waar

B Boolean variabele
en alle boolean gedeclareerde functies zoals
en wachtrij.EMPTY uit 6.4.1

Voorbeeld:
B := ((x < = 5) AND (P =/= NONE)) EQU TRUE

N.B. Binnen alle expressies kan men d.m.v. haakjes () de gewenste volgorde van uitvoeren aangeven.

6.5.4 Statements

- **Block Statement (BS)**

  Een SIMULA programma als geheel bestaat uit een groot blockstatement. Een blockstatement (BS) heeft de vorm:

  BEGIN d1; d2; ...; dn; s1; s2; ...; sm END

  waarbij di staat voor declaratie i (zie 6.6.2) en sj voor statement j (dit kan weer een blockstatement zijn).
  In SIMULA worden statements en/of declaraties onderling gescheiden door een puntkomma. Voor een END hoeft geen puntkomma geplaatst te worden.

- **Compound statement (CS)**

  Een compound statement is een blockstatement zonder declaraties:

  BEGIN s1; s2; ....; sk END

- **Assignment statement (AS)**

  Een assignment statement bestaat uit de toekenning van een expressie aan een variabele.

  VB := BE de boolean variabele VB wordt boolean expressie BE
  VA := AE de aritmetische variabele VA wordt aritmetische expressie AE
  VO :- OE de object variabele (pointer) VO wordt de object expressie OE
  VC := CE de character variabele VC wordt de character expressie CE

  Binnen SIMULA zijn ook meervoudige toekenningen mogelijk.
  Bijvoorbeeld:

  A := B := C := expressie

  met als effect

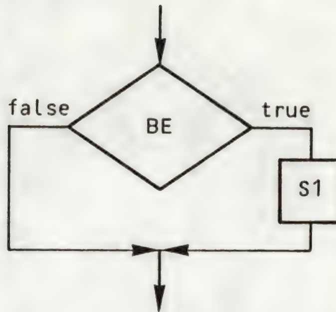
  C := expressie; B := C; A := B.

- **IF statement (IS)**

Het IF statement komt in twee vormen voor:

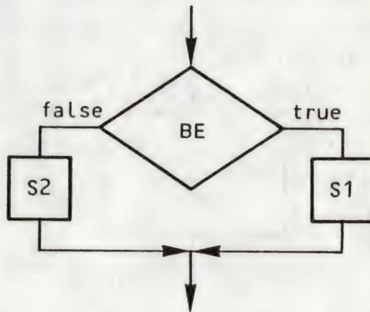
1. IF BE THEN S1 BE : boolean expressie
   S1 : statement

   Dit kan als volgt schematisch voorgesteld worden in de vorm van een stroomdiagram:



2. IF BE THEN S1 ELSE S2 BE : boolean expressie
   S1 : statement
   S2 : statement

   in stroomdiagramvorm:



Er dient op gelet te worden dat er tussen S1 en 'ELSE' nooit een puntkomma mag staan. (Dan wordt ELSE nl. opgevat als het begin van een nieuw, onbekend statement!).

S1 mag niet uit een conditioneel statement bestaan, dus niet alleen IS, WS, of FS.
Bijvoorbeeld:

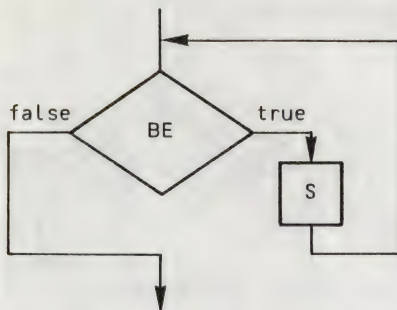
IF BE1 THEN IF BE2 THEN ...

is binnen SIMULA niet toegestaan.

- **WHILE statement (WS)**

Het WHILE statement dient voor implementatie van programmaloops en heeft de volgende vorm:

WHILE BE DO S   BE: boolean expressie
S: statement

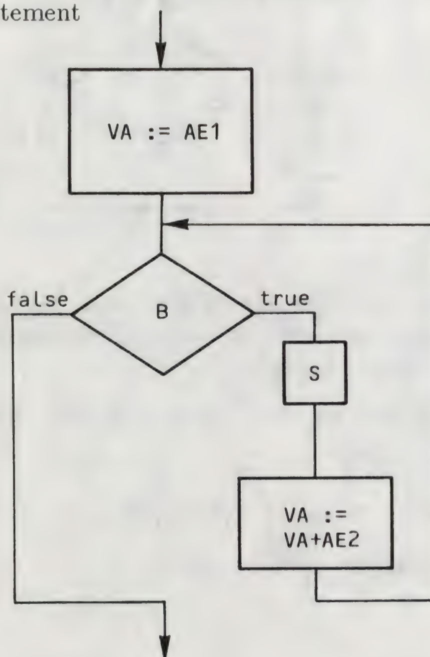


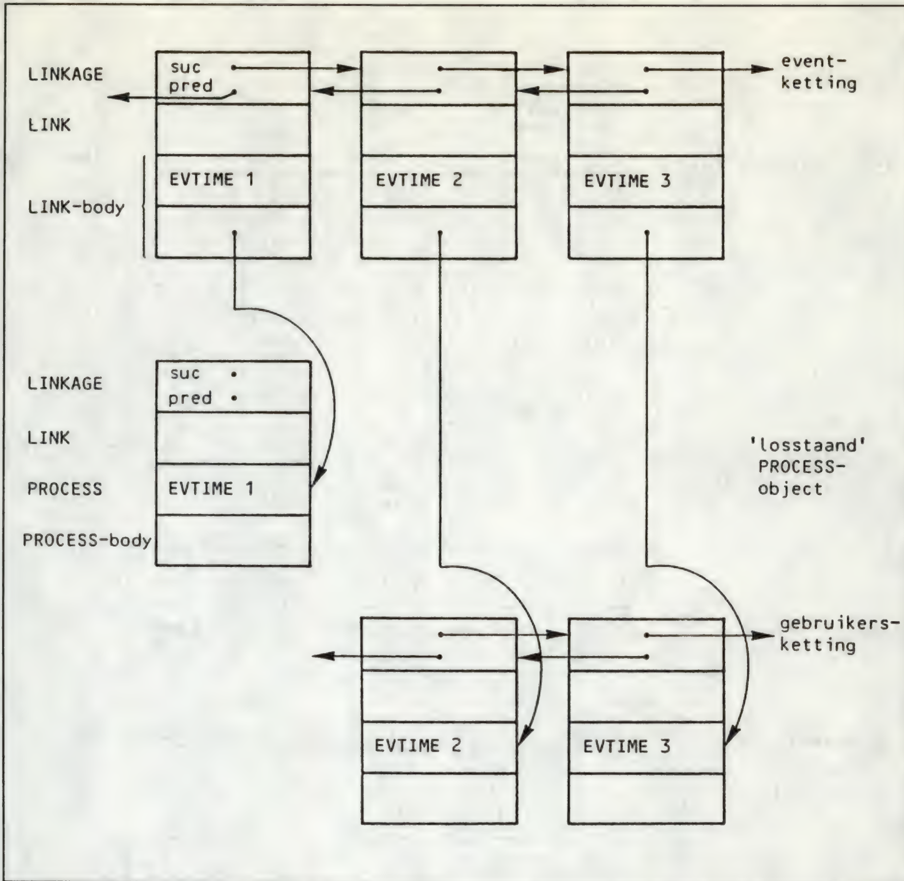
- **FOR statement (FS)**

Het FOR statement dient voor het een vast aantal malen uitvoeren van een statement, en heeft de vorm:

FOR VA := AE1 STEP AE2 UNTIL AE3 DO S

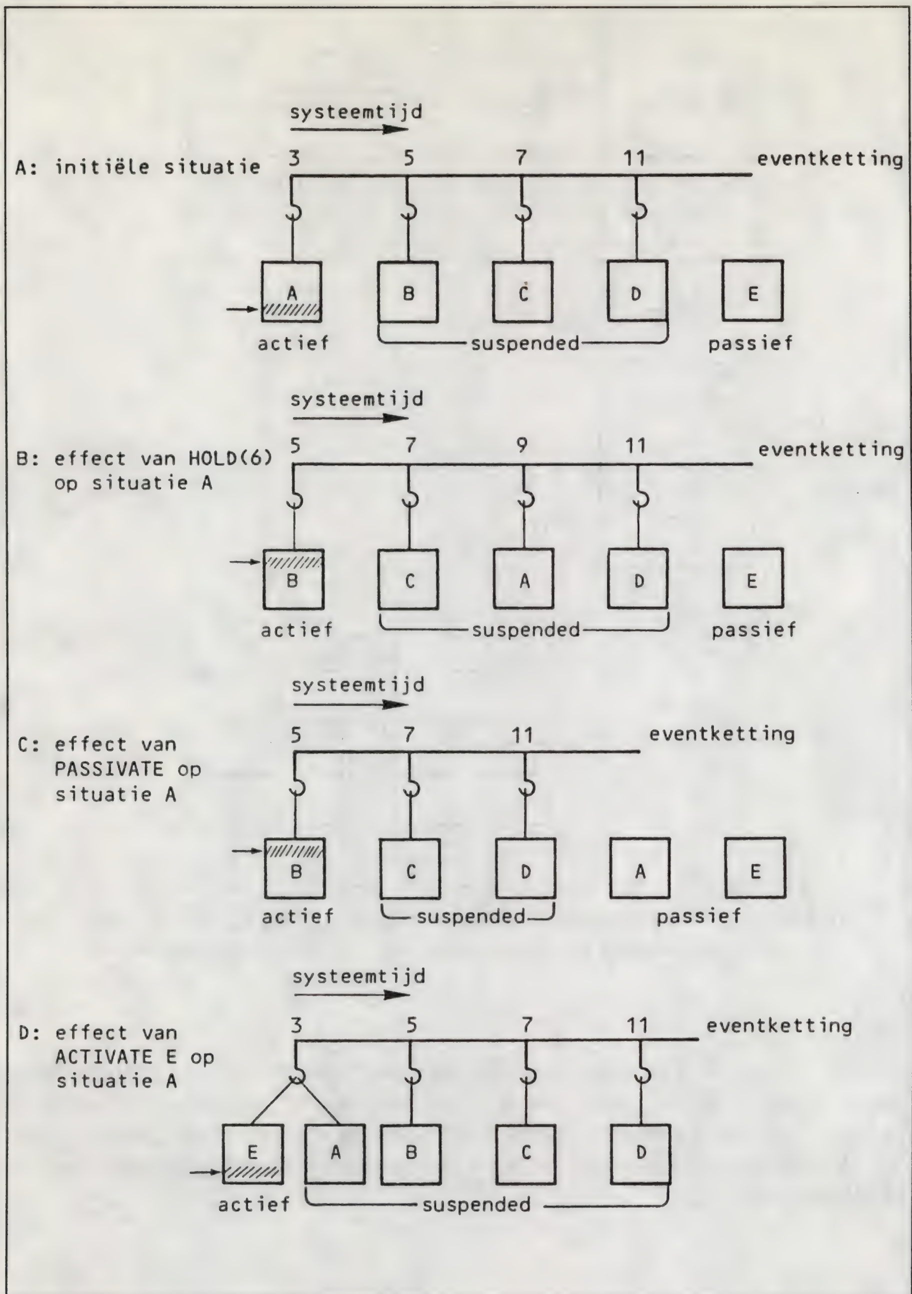
met VA: aritmetische variabele
AE1, AE2, AE3 aritmetische expressies
S: statement





Figuur 6.6: De relaties tussen de eventketting en PROCESS-objecten, die tevens deel uit kunnen maken van gebruikerskettingen

Om de werking van een aantal standaardprocedures m.b.t. PROCESS-objecten toe te lichten zullen we in plaats van figuur 6.6 de meer symbolische figuur 6.7 gebruiken waarin de PROCESS-bodies centraal staan. Hierin wordt het klokmechanisme binnen SIMULA voorgesteld als een ketting met 'haken' (= de eventketting).



Figuur 6.7: Quasi-parallelle processen in SIMULA m.b.v. een eventketting. Het gearceerde gedeelte van het actieve object geeft de positie aan van het statement waarmee de executie van de body van een component (her)start.

- **Procedurestatement (PS)**

  Men mag alle standaard- en zelf gedefinieerde procedures als statement aanroepen, waarbij men wel de parameters, indien aanwezig, van waarden moet voorzien met behulp van expressies. Dus

  procedurenaam(EL)

  indien parameters aanwezig
  EL: expressielijst voor parameters

  of

  procedurenaam

  indien geen parameters aanwezig
  Voorbeeld:

  OUTINT(A,b)

  is de aanroep van de standaardprocedure voor het afdrukken van een variabele A van het type INTEGER waarvoor b posities gebruikt worden.

- **COMMENT statement**

  Een COMMENT statement dient voor het toevoegen van tekst aan een programma welke door de compiler wordt genegeerd.
  Het heeft de volgende vorm:

  COMMENT commentaar . . . ;

  Het einde van het commentaar wordt aangegeven door de eerstvolgende puntkomma. Tevens geldt dat alles wat na een END komt tot aan de eerstvolgende END of puntkomma ook als commentaar opgevat wordt.
  Bijvoorbeeld:

  BEGIN BEGIN S END commentaar END
  BEGIN S; BEGIN S END commentaar; S END

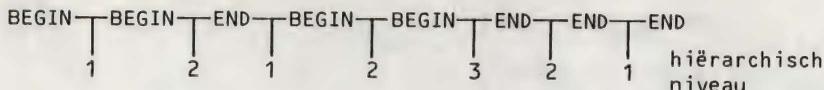
6.5.5 Het bereik (geldigheidsgebied) van variabelen

Iedere gedeclareerde variabele in SIMULA is gedefinieerd en dus bereikbaar zolang het blok waarin deze gedeclareerd is, in uitvoering is. Dus in het programma segment (blok)

BEGIN INTEGER K; BEGIN INTEGER I; S1 END; S2 END

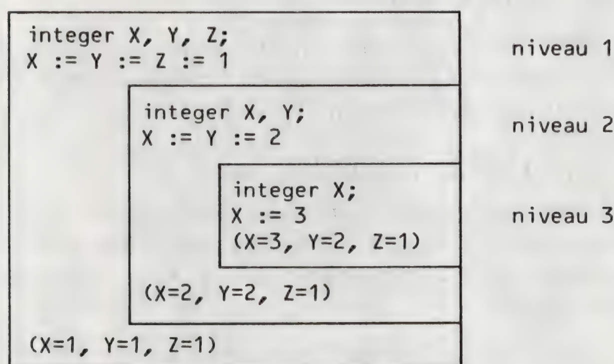
kunnen de variabelen K en I beide in statement S1 gebruikt worden terwijl in statement S2 dit alleen voor variabele K geldt. Een uitzondering op deze regel vormen de locale declaraties in een class object. Deze blijven gedefinieerd zolang er een pointer naar het betreffende object verwijst. Locale variabelen in een object zijn van buiten dat object bereikbaar via de puntnotatie.

Binnen SIMULA zijn variabelen hiërarchisch gerangschikt, waarbij men dieper in de hiërarchie gaat naarmate meer blocks openstaan bij de declaratie. Een block staat open wanneer van dit block wel de BEGIN is geweest maar niet de END. Bijvoorbeeld:



Bij aanroep van een variabele wordt vanaf het niveau van de aanroep naar boven toe alle niveau's doorzocht naar de eerste declaratie van een variabele met deze naam. Wordt geen declaratie gevonden dan resulteert dit in een foutmelding.

Wordt wel een declaratie gevonden dan wordt de huidige waarde van die betreffende variabele genomen. Dit heeft dus tot gevolg dat van hiërarchisch dubbel gedeclareerde variabelen slechts die van het diepste niveau bereikbaar is.
Ter illustratie:



Tussen haakjes staan de waarden van de variabelen op dat niveau

```
BEGIN INTEGER X, Y, Z; X := Y := Z := 1;
      BEGIN INTEGER X, Y; X := Y := 2;
            BEGIN INTEGER X; X := 3
            END
      END
END
```

Vermijd zoveel als mogelijk het declareren van variabelen met dezelfde naam!

Voor procedures, functies en classes geldt dat de binnen deze gedeclareerde variabelen 'voorrang hebben' op de erbuiten gedefinieerde variabelen.

Bij gebruik van de puntnotatie voor classobjecten wordt niet meer hiërarchisch gezocht. Komt de gevraagde variabele niet voor in het object record dan volgt er een foutmelding.

6.5.6 Standaard procedures en functies ten behoeve van in- en uitvoer

De volgende standaard input functies zijn beschikbaar:

ININT	getal zonder decimale punt, bijv. 12
INREAL	getal met decimale punt, bijv. 12.3

Deze functies leveren bij aanroep de waarde van het ingelezen getal.
Bijvoorbeeld:

```
I:=ININT;
```

SIMULA kent de volgende standaard output procedures:

OUTIMAGE	op een nieuwe regel verder gaan
OUTINT(AE,n)	uitprinten integer variabele AE waarbij n posities gebruikt worden
OUTFIX(AE,M,W)	uitprinten real variabele waarbij W posities gebruikt worden (waarvan 1 positie wordt gebruikt voor het afdrukken van de decimale punt) met M symbolen achter de decimale punt
OUTCHAR(CE)	uitprinten een karakter
OUTTEXT("tekst")	uitprinten tekst

6.6 Statistische functies

Binnen SIMULA zijn standaard functies beschikbaar om trekkingen uit verschillende verdelingen te genereren.

Bij de implementatie van deze functies wordt gebruik gemaakt van een random generator.

Een random generator kan worden voorgesteld als een procedure f die uitgaande van een integer getal een ander 'willekeurig' integer getal berekent. Een serie random getallen krijgt men dan als volgt:

UO	startwaarde
U1=f(UO)	1e getal
U2=f(U1)	2e getal
⋮	
Un=f(Un-1)	ne getal

Uitgaande van een gegeven startwaarde levert de random generator steeds dezelfde serie random getallen. Hierdoor is het mogelijk om het gedrag van

modellen onder verschillende condities (bijvoorbeeld verschillende parameterwaarden van het model) te vergelijken, waarbij verschil in modeluitkomsten *niet* wordt veroorzaakt door andere trekkingen uit een bepaalde verdeling. Voor het reproduceerbaar genereren van een serie random getallen bevatten de standaard statistische functies de parameter U. Deze parameter U dient enerzijds als invoergetal voor de randomgenerator, anderzijds als opslagplaats voor het nieuw berekende 'random' getal. Binnen SIMULA wordt het type van een parameter die zowel voor invoer als voor uitvoer dient, gedeclareerd met behulp van NAME. Daarnaast moet ook nog apart gedeclareerd worden tot welk type U behoort. Door een andere startwaarde voor U te kiezen (waarbij U niet negatief mag zijn) kan men een andere reeks random getallen en zo dus trekkingen genereren.

Hieronder volgen enkele veel gebruikte standaard beschikbare statistische functies.

1. BOOLEAN PROCEDURE **DRAW**(A,U); NAME U; REAL A; INTEGER U;
   levert indien:

   0 <= A <= 1 : de waarde TRUE met kans A, de waarde FALSE met kans 1-A.
   A < 0 : de waarde FALSE
   A > 1 : de waarde TRUE

2. INTEGER PROCEDURE **RANDINT**(A,B,U); NAME U; INTEGER A,B,U;
   levert indien:

   A <= B : met gelijke kans één van de waarden A, A+1, ..., B-1, B.
   A > B : een foutmelding.

3. REAL PROCEDURE **NEGEXP**(A,U); NAME U; REAL A; INTEGER U;
   levert indien:

   A > 0 : een trekking uit een negatief exponentiële verdeling met gemiddelde 1/A (dus met intensiteit A).
   A <= 0 : een foutmelding.

4. INTEGER PROCEDURE **POISSON**(A,U); NAME U; REAL A; INTEGER U;
   levert indien:

   A >= 0 : een trekking uit een Poissonverdeling met parameter A.
   A < 0 : de waarde nul.

5. REAL PROCEDURE **ERLANG**(A,B,U); NAME U; REAL A,B; INTEGER U;
   levert indien:

   A > 0 en B > 0 : een trekking uit een Erlangverdeling met gemiddelde 1/A en standaard deviatie 1/(A * B).
   A <= 0 of B <= 0: een foutmelding.

6. REAL PROCEDURE **UNIFORM**(A,B,U); NAME U; REAL A,B; INTEGER U;
levert indien:

A < B : een trekking x uit een uniforme verdeling in het bereik A =< x < B.
A >= B : een foutmelding.

7. REAL PROCEDURE **NORMAL**(A,B,U); NAME U; REAL A,B; INTEGER U;
levert indien:

B >= 0.0 : een trekking uit een normale verdeling met gemiddelde A en standaard deviatie B.
B < 0.0 : een foutmelding.

6.7 De implementatie van een procesbeschrijving voor het tankstation

Uitgangspunt hierbij is figuur 2.14. Hieronder volgen per componentklasse de bijbehorende attributen en procesbeschrijving (activiteitenlijst).

1. Componentklasse aankomstgenerator
1.1 Attributen
gemtat (gemiddelde tussenaankomsttijd)
1.2 Activiteitenlijst:

```
WHILE TRUE
DO    bepaal aankomsttijdstip volgende auto;
      wacht tot aankomsttijdstip volgende auto;
      creëer en activeer nieuwe auto;
      laat waarnemer aantal aangekomen auto's,
      naankomst, aanpassen.
OD;
```

2. Componentklasse auto
2.1 Attributen.
aankomsttijd, bedieningstijd,
2.2 Activiteitenlijst:

```
noteer aankomsttijd, bepaal bedieningstijd,
IF rijlengte < maxrijl
THEN ga in wachtrij
      IF bediende (pomp) = vrij THEN activeer bediende
ELSE rij door;
      laat waarnemer aantal doorgereden auto's,
      ndoorrijder, aanpassen;
```

3. Componentklasse pompbediende.
3.1 Attributen:
vrij (het vrij zijn van de pompbediende),
klant (referentie naar de te bedienen auto)

3.2 Activiteitenlijst:
WHILE TRUE
DO IF wachtrij leeg THEN wacht op aankomst auto
in wachtrij; neem eerste auto uit wachtrij;
laat waarnemer wachttijd auto noteren (somwt);
wacht tot einde bedieningstijd;
auto vertrekt; laat waarnemer aantal bediende auto's,
nbediend, aanpassen;
OD;

4. Componentklasse rapportage(waarnemer)
4.1 Attributen:
naankomst, ndoorrijder, nbediend, somwt, somkwt
4.2 Activiteitenlijst:
Diverse waarneemprocedures om de gewenste metingen
uit te kunnen voeren;
Initialisatieprocedure attribuutwaarden;

Voor het kunnen uitvoeren van de simulatie zijn de volgende componenten nodig:

- 1 aankomstgenerator van de componentklasse aankomstgenerator,
- een nog onbekend aantal auto's van de componentklasse auto (er zal een dag van 24 uur gesimuleerd worden)
- 1 bediende van de componentklasse pomp(bediende),
- 1 waarnemer van de componentklasse rapportage
- 1 wachtrij van de systeemklasse HEAD.

Het simulatieproces verloopt globaal als volgt:

- initialisatie (creatie/activatie diverse componenten)
- wachten tot de tijd nodig voor het simuleren van 1 werkdag van 24 uur voorbij is (aangegeven door middel van het rondje in de procesbeschrijving van de besturing in figuur 2.14)
- rapportage van de gewenste output.

Uitgaande van het bovenstaande komen we tot de volgende implementatie van het tankstation van voorbeeld 2.1 in SIMULA.

Voorbeeld 6.12: *Het SIMULA programma voor het tankstation*

```
BEGIN COMMENT tankstation;
      SIMULATION
      BEGIN COMMENT Hieronder volgen alle declaraties;

            REAL gemtat, og, bg;
            INTEGER simulduur,maxrijl,startw;

            PROCESS CLASS generator(gemtat);REAL gemtat;
            BEGIN WHILE TRUE DO
                  BEGIN HOLD(NEGEXP(1/gemtat, startw));
                        NEW auto;
                        waarnemer.aankomst
                  END
            END***generator***;

            LINK CLASS auto;
            BEGIN REAL aankomsttijd, bedieningstijd;
                  REAL PROCEDURE wachttijd;
                       BEGIN wachttijd:=Time-aankomsttijd END;
                  aankomsttijd:=Time;
                  bedieningstijd:=UNIFORM(og, bg, startw);
                  IF wachtrij.CARDINAL<maxrijl THEN
                  BEGIN INTO(wachtrij);
                        IF bediende.vrij  THEN ACTIVATE bediende
                  END
                  ELSE
                  waarnemer.doorrijder;
            END***auto***;

            PROCESS CLASS pompbediende;
            BEGIN REF(auto) klant;
                  BOOLEAN vrij;
                  vrij:=TRUE;
                  WHILE TRUE DO
                  BEGIN IF wachtrij.EMPTY THEN PASSIVATE;
                        klant :- wachtrij.FIRST;
                        klant.OUT;
                        vrij:=FALSE;
                        waarnemer.noteerwt(klant.wachttijd);
                        HOLD(klant.bedieningstijd);
                        waarnemer.bediend;
                        vrij:=TRUE
                  END
            END***pompbediende***;

            CLASS rapportage;
            BEGIN INTEGER naankomst,ndoorrijder,nbediend;
```

```
        REAL somwt,somkwt;
        PROCEDURE aankomst;
             BEGIN naankomst:=naankomst+1 END;
        PROCEDURE doorrijder;
             BEGIN ndoorrijder:=ndoorrijder+1 END;
        PROCEDURE noteer(wt);REAL wt;
             BEGIN somwt:=somwt+wt;
                   somkwt:=somkwt+wt*wt;
             END;
        PROCEDURE bediend;
             BEGIN nbediend:=nbediend +1 END;
        REAL PROCEDURE spreiding;
             BEGIN IF nbediend>1 THEN
                      spreiding:=SQRT((somkwt-
                      somwt*somwt/nbediend)/(nbediend-1))
                   ELSE spreiding:=-1
             END;
        PROCEDURE report;
             BEGIN OUTIMAGE;
                   OUTTEXT("aantal aankomsten            :");
                   OUTINT(naankomst,6);OUTIMAGE;
                   OUTTEXT("aantal doorrijders           :");
                   OUTINT(ndoorrijder,6);OUTIMAGE;
                   OUTTEXT("aantal auto's bediend        :");
                   OUTINT(nbediend,6);OUTIMAGE;
                   OUTTEXT("percentage doorrijders       :");
                   OUTFIX(100*ndoorrijder/naankomst,2,7);
                   OUTTEXT("%");OUTIMAGE;OUTIMAGE;
                   OUTTEXT("gemiddelde wachttijd         :");
                   IF waarnemer.nbediend>0 THEN
                   OUTFIX(somwt/nbediend,2,7)
                   ELSE OTTEXT("GEEN BETEKENIS");
                   OUTIMAGE;
                   OUTTEXT("spreiding                    :");
                   OUTFIX(spreiding,2,7);OUTIMAGE
             END;

        naankomst:=ndoorrijder:=nbediend:=0;
        somwt:=somkwt:=0.0;
  END***rapportage***;

  PROCEDURE initialiseer;
  BEGIN gemtat:=4;og:=2;bg:=5;
        startw:=1785392;simulduur:=24*60;maxrijl:=3;
        waarnemer :- NEW rapportage;
        bediende :- NEW pompbediende;
        wachtrij :- NEW HEAD;
        gen :- NEW generator(gemtat)
  END;
```

```
        REF(pompbediende) bediende;
        REF(rapportage) waarnemer;
        REF(HEAD) wachtrij;
        REF(generator) gen;

        COMMENT Hieronder volgt het hoofdprogramma MAIN;

        BEGIN initialiseer;
              ACTIVATE bediende;
              ACTIVATE gen;
              HOLD(simulduur);
              waarnemer.report;
              OUTIMAGE
        END
    END
END
```

```
@execute pomp.sim
SIMULA:  POMP
LINK:    Loading
[LNKXCT  POMP execution]

aantal aankomsten              :   347
aantal doorrijders             :    24
aantal auto's bediend          :   322
percentage doorrijders         :   6.92%

gemiddelde wachttijd           :   3.40
spreiding                      :   3.24


2 garbage collection(s) in 79 ms

End of SIMULA program execution.
CPU time: 3.01  Elapsed time: 5.63
```

Opmerking:
Zoals reeds opgemerkt in hoofdstuk 2 kan eenzelfde systeem op geheel verschillende wijzen met een procesbeschrijving worden weergegeven. Dit hangt er met name vanaf of men het systeem meer vanuit de tijdelijke componenten (klant, produkt) of meer vanuit de vaste componenten (bediende, machine, wachtrij) beschouwt. In het eerste geval ontstaat een uitgebreide beschrijving van de tijdelijke component, waarin als het ware zijn hele gang door het systeem beschreven wordt. De beschrijvingen van de vaste componenten blijven heel kort: het enige wat voor deze van belang is, is of zij bezet of vrij zijn (klant is slim, bediende dom). In het andere geval wordt van elke vaste component volledig

beschreven wat er gebeurt als een tijdelijke component passeert. De beschrijving van de tijdelijke component wordt dan heel beperkt: het enige wat nog van belang is, is zijn 'capaciteitsbehoefte' (klant is dom, bediende slim).

Bij implementatie in SIMULA blijkt in de praktijk dat deze laatse opvatting veelal tot de eenvoudigste programma's leidt.

6.8 Opgaven

1 De eigenaresse van het tankstation van voorbeeld 2.1 vraagt zich af of het de moeite zou lonen een smeerplaats te openen en hiervoor parttime iemand in dienst te nemen.
Verder zou ze de wachtruimte voor zowel de pomp als de smeerplaats 'oneindig' groot willen maken.
Uit een enquete onder haar klanten weet ze dat van elke 10 auto's er gemiddeld 7 alleen bezine nodig hebben en 3 tevens hun auto zouden willen laten smeren.
Ze wil nu weten:

- het aantal auto's dat dan per dag zou komen tanken;
- de gemiddelde doorlooptijd met betrekking tot tanken voor 1 auto;
- het aantal auto's dat dan per dag gesmeerd zou worden;
- de gemiddelde doorlooptijd met betrekking tot het smeren van 1 auto.

Een adviesburo/student heeft in dat kader het onderstaande simulatieprogramma gemaakt.
Verifieer de juistheid van dit programma. □

```
BEGIN
     SIMULATION
     BEGIN
          PROCESS CLASS aankgen;
          BEGIN
               WHILE TRUE DO
               BEGIN
                    HOLD(NEGEXP(1/mu,u1));
                    IF DRAW(0.7,u2) THEN
                       ACTIVATE NEW auto(Time)
                    ELSE
                       ACTIVATE NEW smeerauto(Time);
               END
          END;

          PROCESS CLASS auto(aank); REAL aank;
          BEGIN
               INTO(pomprij);
               IF pomp.vrij THEN ACTIVATE pomp AFTER CURRENT;
               PASSIVATE;
               OUT;
               HOLD(UNIFORM(a,b,frand));
               piet.noteer(TIME-aank);
               ACTIVATE pomp
          END;
```

```
            auto CLASS smeerauto;
            BEGIN
                  REAL start;
                  start:=TIME;
                  INTO(smeerrij);
                  IF smeerkees.vrij THEN
                     ACTIVATE smeerkees AFTER CURRENT;
                  PASSIVATE;
                  OUT;
                  HOLD(UNIFORM(c,d,srand));
                  piet.smeer(TIME-start);
                  ACTIVATE smeerkees;
            END;

            PROCESS CLASS bediende(rij); REF(HEAD) rij;
            BEGIN
                  BOOLEAN vrij;
                  vrij:=TRUE;
                  WHILE TRUE DO
                  BEGIN
                        IF NOT rij.EMPTY THEN
                        BEGIN
                              ACTIVATE rij.FIRST;
                              vrij:=FALSE
                        END;
                        PASSIVATE;
                        vrij:=TRUE
                  END
            END;

            CLASS waarnemer;
            BEGIN
                  PROCEDURE noteer(dltpomp); REAL dltpomp;
                  BEGIN
                        somdpomp:=somdpomp+dltpomp;
                        npomp:=npomp+1
                  END;
                  PROCEDURE smeer(dltsme); REAL dltsme;
                  BEGIN
                        somdsme:=somdsme+dltsme;
                        nsme:=nsme+1
                  END;
                  PROCEDURE schrijf;
                  BEGIN
                        OUTTEXT("run        ");
                        OUTINT(i,2);OUTIMAGE;
                        OUTTEXT("aantal auto's bij de pomp ");
                        OUTINT(npomp,3);OUTIMAGE;
                        OUTTEXT("gem. dlt bij de pomp      ");
                        OUTFIX(somdpomp/npomp,2,7);OUTIMAGE;
                        OUTTEXT("aantal auto's bij smeren  ");
                        OUTINT(nsme,3);OUTIMAGE;
                        OUTTEXT("gem. dlt bij smeren       ");
                        OUTFIX(somdsme/nsme,2,7);OUTIMAGE;
                  END;
                  INTEGER npomp, nsme;
                  REAL somdpomp, somdsme;
                  somdsme:=somdpomp:=0.0;
                  nsme:=npomp:=0
            END;

            REAL mu,a,b,c,d; INTEGER frand,srand,u1,u2,i;
            REF(HEAD) pomprij,smeerrij;
            REF(bediende) pomp,smeerkees;
            REF(waarnemer) piet;

            mu:=4; a:=2; b:=5; c:=4; d:=7;
            frand:=1254; srand:=1876; u1:=12456; u2:=574686;
            pomprij:- NEW HEAD; smeerrij:- NEW HEAD;
            pomp:- NEW bediende(pomprij);
            smeerkees:- NEW bediende(smeerrij);
            piet:- NEW waarnemer;

            ACTIVATE pomp;
            ACTIVATE smeerkees;
            ACTIVATE NEW aankgen;
            FOR i:=1 STEP 1 UNTIL 10 DO
            BEGIN
                  HOLD(8*60);
                  piet.schrijf;
                  piet:- NEW waarnemer;
            END

      END
END
```

2 *Alhoewel SIMULA door haar object-georiënteerde opzet met name geschikt is voor de implementatie van procesbeschrijvingen, hebben we deze opgave opgenomen 'ter illustratie' van de implementatie van een eventbeschrijving in SIMULA.
We zullen daarbij uitgaan van de uitwerking van opgave 2 van hoofdstuk 2. Doe dit aan de hand van de volgende stappen (overeenkomstig 7.2):

8. a. Geef de attributen die voor de beschrijving van de verschillende eventklassen van belang zijn. (Houdt hierbij rekening met de 'waarnemer').
   b. Vervalt hier.
   c. Geef zonodig de eventbeschrijving in pseudocode.
9. Kwantificeer het conceptueel model.
10. Schrijf het SIMULA programma. □

3 Schrijf een SIMULA programma aan de hand van de procesbeschrijving verkregen via de uitwerking van opgave 3 van hoofdstuk 2. Volg hierbij de stappen 8, 9 en 10 van werkwijze 7.2, te weten:

8. a. Geef de attributen per componentklasse (nu inclusief die van de waarnemer).
   b. Geef het aantal benodigde componenten per klasse.
   c. Geef zodanig de procesbeschrijving in pseudocode.
9. Kwantificeer het conceptueel model.
10. Schrijf het SIMULA programma. □

4 Idem voor opgave 4 uit hoofdstuk 2. □

5 Idem voor opgave 5 uit hoofdstuk 2. □

6 Idem voor opgave 6 uit hoofdstuk 2. □

7 Idem voor opgave 7 uit hoofdstuk 2. □

8 *Idem voor opgave 8 uit hoofdstuk 2. □

9 Idem voor opgave 9 uit hoofdstuk 2. □

Hoofdstuk 7

Uitvoerig simulatievoorbeeld in SIMULA

7.1 Inleiding

In dit hoofdstuk wordt een wat moeilijker probleem dan dat van het tankstation uitgewerkt. Uitgaande van de probleemdefinitie wordt een procesbeschrijving gespecificeerd en vervolgens geïmplementeerd met behulp van SIMULA.

In 7.2 zal eerst de hierbij gebruikte werkwijze, gebaseerd op figuur 1.2 en uitgewerkt in de daarop volgende hoofdstukken, in een 13-tal stappen worden opgedeeld. Vervolgens zal in 7.3 het voorbeeld volgens de gegeven werkwijze worden uitgewerkt.

7.2 De werkwijze

In figuur 1.2 is globaal aangegeven volgens welke stappen een probleem met behulp van computersimulatie kan worden opgelost. Voor de procesbeschrijving is de methode/techniek van uitwerken expliciet aangegeven in het volgende stappenplan. Dit stappenplan geldt ook voor de event- en activiteitenbeschrijving, met dien verstande dat stap 6 tot en met 8 enigszins aangepast moeten worden. (Voor de eventbeschrijving is de aanpassing opgenomen in opgave 2.9.2).

1. Formuleer de doelstelling van het onderzoek zo concreet als mogelijk. Stel hierbij de gewenste betrouwbaarheid en nauwkeurigheid van de onderzoeksresultaten vast.
2. Bepaal de in het kader van de doelstelling relevante uitgangs-, stuur-, omgevings- en toestandsvariabelen.
3. Geef een grafische weergave van het conceptueel model.
4. Geef een analytische beschouwing van het gegeven probleem. (Zodat er enerzijds meer inzicht wordt verkregen in de probleemsituatie, anderzijds kunnen de resultaten later gebruikt worden voor een modelvalidatie).
5. a Bepaal per componentklasse de bijbehorende 'activiteiten'.
   b Maak hiervan een doorsnede en som alle activiteitsklassen op.
   c Geef per activiteitsklasse de bijbehorende veranderingen van de toestandsvariabelen.
6. Wijs de activiteitsklassen eenduidig toe aan de componentklassen.

7. Teken het stroomschema van de procesbeschrijvingen van de conponentklassen (zie 2.6).
8. a Geef de attributen per componentklasse (nu inclusief die van de waarnemer).
   b Geef het aantal benodigde componenten per klasse.
   c Geef zonodig de procesbeschrijving in pseudocode.
9. Kwantificeer het conceptueel model.
10. Schrijf het (SIMULA) programma.
11. Bepaal de duur van de aanloopverschijnselen, het aantal benodigde subruns en de lengte ven één subrun ondermeer op basis van de resultaten van proefrun(s). Vergelijk de zo verkregen simulatie resultaten met die verkregen met behulp van de in 4 gegeven analytische beschouwing (deel van de validatiefase).
12. Ontwerp een serie simulatie-experimenten en voer deze uit.
13. Formuleer de conclusies.

N.B. Indien de opgaven zoals weergegeven aan het eind van dit hoofdstuk in een onderwijsomgeving worden gebruikt, is het raadzaam dat tussen stap 9 en 10 een tussentijdse bespreking plaatsvindt.

7.3 Het havenprobleem

In een haven heeft men te maken met twee typen schepen, namelijk kleine en grote schepen (afhankelijk van het tonnage).

Kleine schepen komen binnen volgens een negatief exponentiële verdeling met een verwachte tussenaankomsttijd van 5,5 uur.

Grote schepen komen binnen volgens een negatief exponentiële verdeling met een verwachte tussenaankomsttijd van 6,7 uur.

Kleine schepen worden gelost aan kade 1 voor kleine schepen. De tijd voor het lossen van een klein schip is uniform verdeeld tussen 3 en 7 uur.

Grote schepen worden gelost aan kade 2 voor grote schepen. De lostijd voor een groot schip is uniform verdeeld tussen 2 en 8 uur.

Aan kade 1 kan, indien deze geheel vrij is en er geen kleine schepen in de haven zijn, een groot schip gelost worden. Het lossen duurt dan echter 1,5 maal zo lang als bij het lossen aan kade 2. Omgekeerd kan ook 1 klein schip aan kade 2 gelost worden indien deze vrij is en er geen grote schepen in de haven zijn. De lostijden worden dan 2 keer zo groot als aan kade 1 het geval zou zijn.

De queuediscipline is KBE (kortste bewerkingstijd eerst).

De havendirectie vraagt zich af, of het niet beter zou zijn kade 1 voor grote schepen uit te sluiten en kade 2 voor kleine schepen uit te sluiten. Tevens zou de directie graag weten of de queuediscipline FIFO (first in first out) niet meer geëigend is voor haar haven.
Welk advies moet aan de havendirectie worden gegeven?

Uitwerking

Stap 1: *Doelstelling van het onderzoek*

Uit de gegeven probleemstelling leiden we de volgende doelstelling af: Bepaal het verschil tussen de verwachte gemiddelde doorlooptijden van de schepen voor de twee situaties voor de twee queuedisciplines (zowel voor de grote als de kleine schepen) met een betrouwbaarheid van 95% en een onnauwkeurigheid van 10%.

Stap 2: *Variabelen*

Uit de doelstelling volgt dat de gemiddelde doorlooptijden bepaald moeten worden. Dit zijn dus uitgangsvariabelen die door de 'waarnemer' aangeleverd moeten worden waartoe deze registraties in het gesimuleerde systeem moet verrichtten (zie figuur 2.3). Tevens hebben we te maken met de uitgangsvariabele van het reële systeem, te weten: vetrekkende schepen.

Uit de probleemstelling volgt dat we te maken hebben met 2 stuurvariabelen namelijk de kade toewijzingsstrategie en de queuediscipline.

De omgeving komt tot uiting in de variabelen: tussenaankomsttijd, bedieningstijd en soort schip.

De toestandsvariabelen zijn in dit geval: het aantal wachtende schepen van elke soort en het al dan niet vrij zijn van kade 1 en kade 2.

Stap 3: *Conceptueel model*

Het conceptueel model wordt vastgelegd op de wijze zoals aangegeven in 2.3. Dat wil zeggen dat achtereenvolgens

- de systeemgrens,
- de vaste componentklassen,
- de tijdelijke componentklassen en
- de relaties tussen de componenten moeten worden vastgesteld.

De systeemgrens is in dit voorbeeld duidelijk.

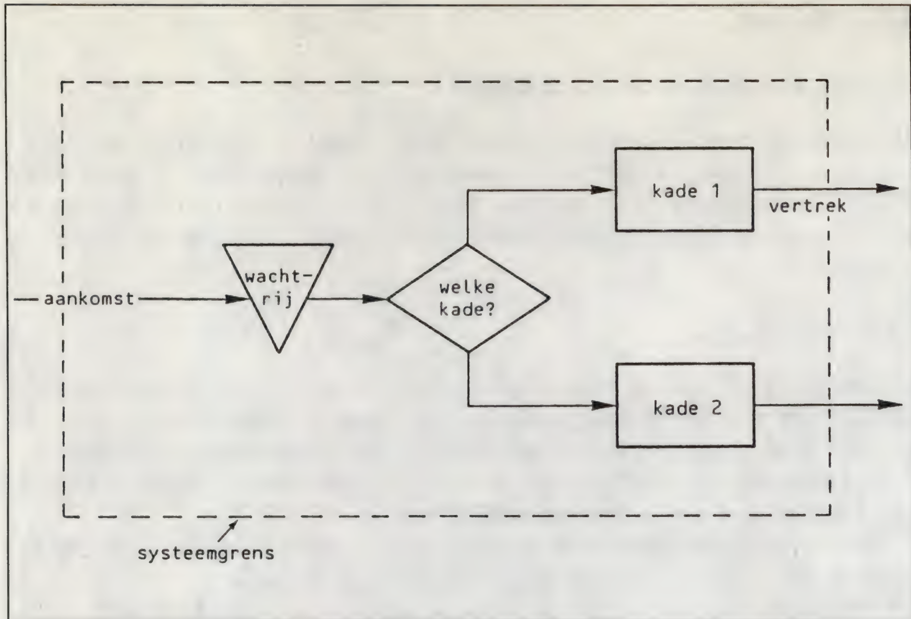
De vaste componentklassen zijn:

- kade (1 klasse, 2 componenten: kade 1 en kade 2),
- wachtrij.

De tijdelijke componentklasse is:

- schip (1 klasse, waarbinnen onderscheid gemaakt wordt tussen kleine en grote schepen).

In onderstaande figuur worden de systeemgrens en de relaties tussen de componenten weergegeven.



Figuur 7.1 Grafische weergave van het conceptueel model van de haven

Stap 4: *Analytische beschouwing*

Het doel van de analytische beschouwing is om langs analytische weg inzicht te krijgen in het gedrag van het systeem. Deze beschouwing zal zoveel mogelijk kwantitatief van aard zijn.

Met betrekking tot het havenprobleem zullen we eerst de nieuwe situatie beschouwen omdat deze het eenvoudigst is.

Kwantitatieve beschouwing nieuwe situatie.

De kleine schepen worden alleen aan kade 1 gelost, de grote schepen alleen aan kade 2.

Een probleem bij deze beschouwing is dat voor de gebruikte wachtrijdiscipline KBE geen analytische resultaten beschikbaar zijn. We zullen daarom eerst de berekeningen uitvoeren voor de eenvoudigere prioriteitsregel FIFO en vervolgens hieruit de resultaten voor KBE schatten.

Bij de nu volgende analyse wordt verondersteld dat de lezer beschikt over enige basiskennis op het gebied van wachtrijtheorie (Wagner 1975).

Kade 1 (FIFO):

Algemeen geldt:

$E(\underline{dlt}) = E(\underline{w}) + E(\underline{v})$, waarin:
$E(\underline{dlt})$ = verwachtingswaarde van de doorlooptijd
$E(\underline{w})$ = verwachtingswaarde van de wachttijd
$E(\underline{v})$ = verwachtingswaarde van de bedieningstijd

In de nieuwe situatie geldt (Pollaczek, zie Wagner 1975):

$$\frac{E(\underline{w})}{E(\underline{v})} = \frac{r}{2(1-r)}(1 + C_{SV}^2), \quad \text{waarin}$$

r = bezettingsgraad van een kade (dit is de verhouding tussen de aankomstintensiteit en de maximaal mogelijke bedieningsintensiteit)
C_{SV} = variatiecoëfficient van de kansverdeling van de lostijden van schepen aan een kade (is gelijk aan standaarddeviatie gedeeld door gemiddelde).

Voor kade 1 geldt nu:

$$C_{SV} = \frac{\sigma_V}{\mu_V} = 0,23$$

Immers de bedieningstijd is uniform verdeeld tussen 3 en 7 zodat

$$\mu_V = \frac{(3+7)}{2} = 5; \qquad \sigma_V^2 = \frac{(7-3)^2}{12} = \frac{4}{3}$$

Voor de bezettingsgraad r van kade 1 geldt:

$$r = \frac{\frac{1}{5,5}}{\frac{1}{5}} = 0,909$$

Met behulp van Pollaczek vinden we dan:

$$E(\underline{dlt}) = E(\underline{w}) + E(\underline{v}) = 26,3 + 5 = 31,3$$

Kade 1 (KBE):

Volgens de wachtrijtheorie wordt het gemiddelde aantal wachtende klanten met zowel negatief exponentieel verdeelde binnenkomstintervallen als bedieningstijden bij gebruik van de KBE-prioriteitsregel ongeveer gehalveerd ten opzichte van het gemiddelde aantal klanten in de wachtrij bij gebruik van de FIFO-regel. In onze benadering hebben we echter te maken met een meer deterministische bedieningstijd waardoor het halveringseffect niet volledig zal optreden. We kunnen stellen dat de doorlooptijd zal liggen tussen de bovengrens, gebaseerd op een deterministische bedieningstijd $C_{SV} = 0$; geen halveringseffect) en de ondergrens, gebaseerd op een negatief exponentieel verdeelde bedieningstijd ($C_{SV} = 1$; halveringseffect).
De bijbehorende grenswaarden zijn:

$18,3 \leq E(\underline{dlt}) \leq 31,3$

Stel nu op basis van $C_{SV} = 0,23$ dat het verkortingseffect zeer ruw geschat gelijk is aan:

$0,23 * 0,5 * 26,3 \approx 3,0.$
Dan geldt:
$E(\underline{w}) = 23,3$
$E(\underline{dlt}) = E(\underline{w}) + E(\underline{v}) = 23,3 + 5 = 28,3$

Kade 2 (FIFO):

Op analoge wijze berekenen we voor kade 2:

$r = 0,745$
$E(\underline{w}) = 8,2$
$E(\underline{dlt}) = 13,2$

Kade 2 (KBE):

Bij de berekening van de wachttijd met gebruikmaking van de KBE gaan we hetzelfde te werk als bij kade 1. Dus ruw geschat geldt:

$9,1 \leq E(\underline{dlt}) \leq 13,2$
$E(\underline{w}) = 8,2 - 0,35 * 0,5 * 8,2 = 6,8$
$E(\underline{dlt}) = 6,8 + 5 = 11,8$

Het op basis van de simulatieresultaten te berekenen betrouwbaarheidsinterval van de gemiddelde doorlooptijd zal op zijn minst overlap moeten vertonen met hiervoor aangeduide interval. Een kwantitatieve beschrijving van de oorspronkelijke situatie is te moeilijk. We zullen ons daarom beperken tot een kwalitatieve beschouwing.

Kwalitatieve beschouwing oude situatie.

We willen proberen vast te stellen wat het effect is van de invoering van de nieuwe situatie op de gemiddelde doorlooptijd van de verschillende soorten schepen.

Hiertoe bepalen we achtereenvolgens de kans dat een groot schip aan kade 1 gelost kan worden en de kans dat een klein schip aan kade 2 gelost kan worden.

In de oorspronkelijke situatie kunnen grote schepen, indien er geen kleine schepen in het systeem zijn, aan kade 1 gelost worden. De kans dat er geen kleine schepen in het systeem zijn is:

$1 - r = 0,10.$

Een groot schip kan aan kade 1 gelost worden als zich twee of meer grote schepen in het systeem bevinden. De kans hierop is 0,47 voor een M/D/1 model (zie bijvoorbeeld Hillier & Yu 1981).

Deze kansen suggereren dat grote schepen niet vaak gelost zullen worden aan kade 1. Als het echter toch gebeurt, dan zal de wachtrij kleine schepen langer worden aangezien het grote schip het 'loket' bezet houdt.

We nemen daarom aan dat de verwachte doorlooptijd van de kleine schepen in de oorspronkelijke situatie door het bovengenoemde effekt iets hoger zal zijn dan in de nieuwe situatie en die van de grote schepen iets lager.

Kleine schepen krijgen in de oorspronkelijke situatie een kans op een ligplaats aan kade 2 als er geen grote schepen in het systeem aanwezig zijn.

De kans dat er zich geen grote schepen in het systeem bevinden is gelijk aan $1 - 0{,}75 = 0{,}25$.

Verder moeten dan twee of meer kleine schepen in het systeem aanwezig zijn. Aangezien de bezettingsgraad voor kleine schepen 0.90 is, is de kans hierop voor een M/D/1 systeem 0,75.

Hieruit kunnen we concluderen dat de kans dat kleine schepen gelost worden aan kade 2 beslist niet te verwaarlozen is.

Indien deze situatie zich voordoet zal dit in het algemeen voor de grote schepen wachttijdverhogend werken.

Voor de kleine schepen zal het wachttijdverlagend werken.

Samenvattend kunnen we stellen dat de verwachte doorlooptijd van de grote/kleine schepen in de oorspronkelijke situatie hoger/lager zal liggen dan in de nieuwe situatie.

Stap 5: *Activiteitsklassen*

Om tot een procesbeschrijving van het model te komen, is het nodig om de activiteiten van de vaste en tijdelijke componentklassen vast te stellen (zie 2.3).

Vooruitlopend op de implementatiefase voegen we de vaste componentklasse 'aankomstgenerator' toe (zie figuur 2.14). In het reële systeem bestaat deze component niet, maar wordt bij de computersimulatie gebruikt om ook het aankomstproces na te kunnen bootsen.

a. De 'activiteiten' per componentklasse zijn:

aank.gen.	wachtrij	schip	kade
-genereert aankomst		-schip komt haven binnen	
	-neemt schip op	-gaat in wachtrij staan	
	-schuift schip door	-schuift door	
	-verwijdert schip	-gaat uit wachtrij	-haalt schip uit wachtrij
	-plaatst schip aan kade	-gaat aan kade staan	-plaatst schip aan kade
		-schip wordt gelost	-lost schip
		-verlaat het systeem	-verwijdert het schip.

b. Een doorsnede van deze activiteiten leidt tot de volgende activiteitklassen:
   A1: genereer aankomst schip
   A2: plaats schip in de wachtrij
   A31: haal schip uit de wachtrij
   A32: plaats schip aan de juiste kade
   A4: start lossen schip
   A5: haal schip van de kade en verwijder het uit het systeem
   (A6: schuif het schip door).
   De activiteitsklassen A31 en A32 treden hier altijd in combinatie op. Ze worden daarom verder samen aangeduid als A3.

c. De bijbehorende veranderende toestandsvariabelen zijn:
   A2 ⟶ rijlengte.
   A3 ⟶ rijlengte, kade vrij.
   A5 ⟶ kade vrij.

Stap 6: *Toewijzing activiteitsklassen*

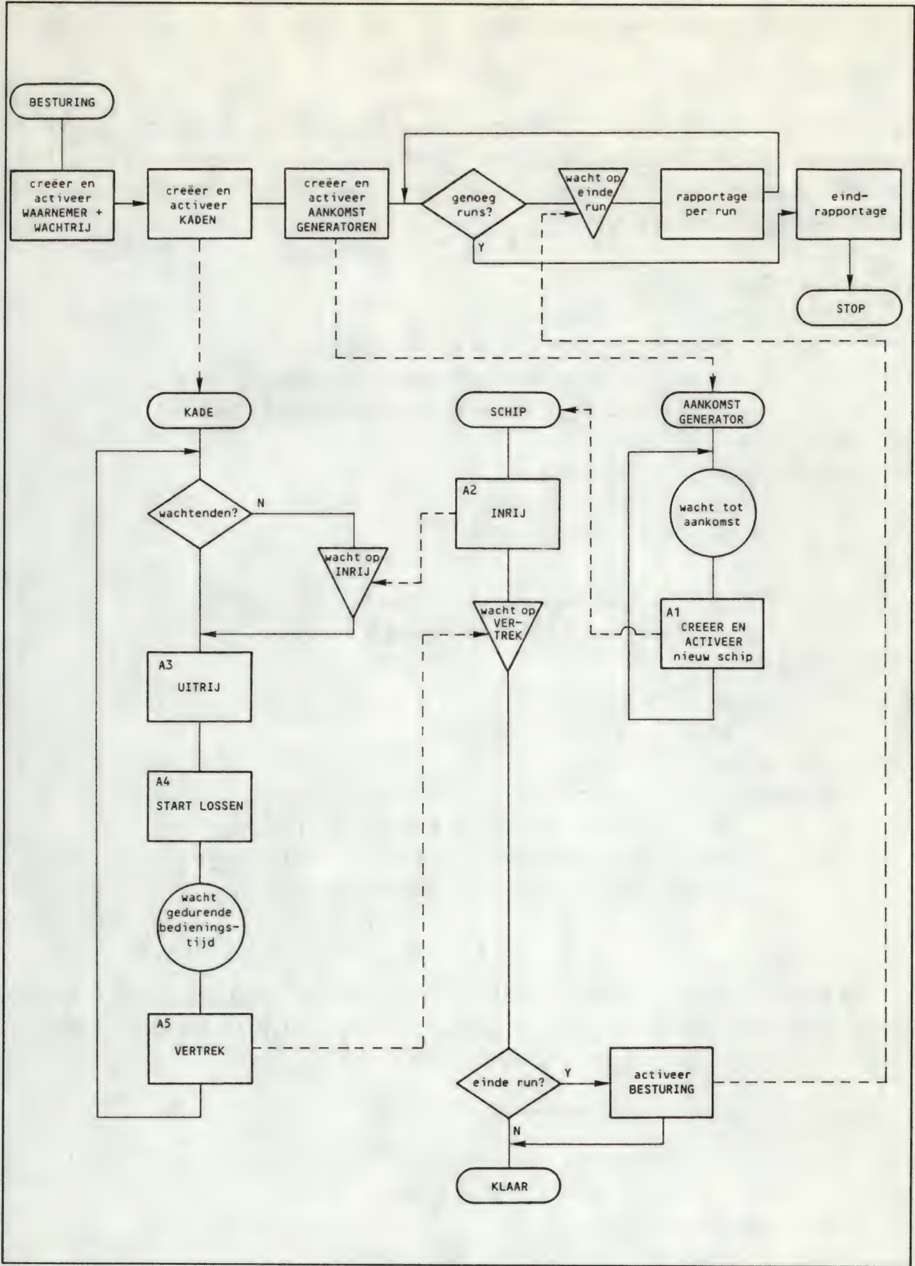
De gevonden activiteitsklassen dienen nu eenduidig toegewezen te worden aan de componentklassen (zie 2.6).

We zullen verder uitgaan van de volgende toewijzing:

Activiteitsklassen	Componentklassen			
	aankomstgenerator	schip	kade	wachtrij
A1	x			
A2		x		
A3			x	
A4			x	
A5			x	
A6				x

Stap 7: *Het stroomschema*

In het navolgende stroomschema is de component wachtrij niet verder uitgewerkt omdat de activiteit 'schuifdoor' automatisch uitgevoerd wordt bij implementatie van een wachtrij binnen SIMULA via een lijst (vergelijk fig. 2.11, 2.12 en 2.14).



Figuur 7.2: Stroomschema haven (In dit schema is de registratie van te rapporteren gegevens niet expliciet aangegeven).

Stap 8: *De procesbeschrijving in pseudo-code*

a. *De attributen per componentklasse:*

De relevante attributen volgen uit de toestandsvariabelen en de overige onder punt 2 genoemde variabelen alsmede uit de gekozen implementatiewijze (gestreefd is naar zo universeel mogelijke klassen, vandaar het attribuut ss bij de verschillende componentklassen).

C1: aankomstgenerator
attributen: ss, schipsoort
gemtat gemiddelde tussenaankomsttijd
og, ondergrens uniforme verdeling lostijden
bg, bovengrens uniforme verdeling lostijden
C2: schip
attributen: ss, schipsoort
ta, aankomsttijd
tb, bedieningstijd
C3: kade
attributen: ss, kade primair bestemd voor schipsoort ss
plaats, geeft aan of er plaats is aan kade ss
C4: wachtrij
attributen: ss, schipsoort
nrij, rijlengte.
C5: waarnemer
attributen: nk, aantal vertrokken kleine schepen
ng, aantal vertrokken grote schepen
gdk, gemiddelde doorlooptijd kleine schepen
gdg, gemiddelde doorlooptijd grote schepen

In verband met het plaatsingsalgoritme van de schepen aan de kaden lijkt het zinvol de twee soorten schepen na aankomst gescheiden te houden. Vandaar twee wachtrijen, één voor grote en één voor kleine schepen.

b. *Aantal benodigde componenten:*

C1: twee: één generator voor kleine schepen,
één generator voor grote schepen.
C2: moet nog bepaald worden
C3: twee: één kade primair voor kleine schepen,
één kade primair voor grote schepen.
C4: twee: één wachtrij voor kleine schepen,
één wachtrij voor grote schepen.
C5: één

c. *De procesbeschrijving in pseudocode:*

```
C1: aankomstgenerator(ss, gemtat, og, bg)

    WHILE TRUE
    DO trek tussenaankomsttijd;
       wacht gedurende tussenaankomsttijd;
       A1 (trek bedieningstijd;
           creëer + activeer schip);
    OD;

C2: schip(ss, ta, tb)

    A2 (plaats schip in wachtrij (ss);
       indien een kade vrij is activeer dan
       indien mogelijk kade (ss) en anders
       kade (notss));
    wacht op A5 (vertrek van het schip uit het systeem);
    laat waarnemen doorlooptijd en vertrek van schip registreren;
    IF (nk+ng)>= subrunlengte THEN activeer MAIN;

C3: kade(ss)

    WHILE TRUE
    DO WHILE rijlengte (ss) > 0 OR rijlengte(notss) > 0
       DO IF rijlengte (ss) > 0
          THEN A3 (haal een schip uit wachtrij(ss) en
                   plaats aan kade(ss))
          ELSE A3 (haal een schip uit wachtrij(notss) en
                   plaats aan kade(ss));
          A4 (start lossen van het schip);
          wacht gedurende bedieningstijd;
          A5 (maak kade(ss) vrij en laat schip vertrekken);
       OD;
       wacht op A2;
    OD;
```

Stap 9: *Kwantificeren van het conceptuele model*

	ss	gemtat	og	bg
aankomstgenerator kl. schepen	k	5,5	3	7
aankomstgenerator gr. schepen	g	6,7	2	8
kade voor kleine schepen	k	-	-	-
kade voor grote schepen	g	-	-	-
wachtrij voor kleine schepen	k	-	-	-
wachtrij voor grote schepen	g	-	-	-
rijdiscipline: KBE				

Stap 10: *Het SIMULA programma*

Achtereenvolgens zullen worden gepresenteerd:

- een lijst met de betekenis van de gebruikte variabelen,
- de source-tekst van het SIMULA programma,
- een korte toelichting op het programma.

Gebruikte variabelen/functies en hun betekenis.

AFKORTING	BETEKENIS
bg	bovengrens van de uniforme verdeling van lostijden
dlt	doorlooptijd van een schip
dummyrij	lege wachtrij (wordt gebruikt om met het hetzelfde programma de oude en nieuwe situatie te kunnen simuleren)
gdg	gemiddelde doorlooptijd voor grote schepen.
gdk	gemiddelde doorlooptijd voor kleine schepen.
gemdlt1(k)	gemiddelde dlt kleine schepen voor subrun k
gemdlt2(k)	gemiddelde dlt grote schepen voor subrun k
gemtat	gemiddelde tussenaankomsttijd schepen
grij	wachtrij voor grote schepen.
i,j,k	hulptellers.
krij	wachtrij voor kleine schepen.
lostijd	de tijd nodig om een schip te lossen
NEGEXP	procedure die getallen genereert uit een negatief exponentiële verdeling.
ng	aantal vertrokken grote schepen tot nu toe
nk	aantal vertrokken kleine schepen tot nu toe
nsa	aantal schepen aanlooprun
nsps	aantal schepen per subrun.
nstop	nsa of nsps
nsubr	aantal subruns.
og	ondergrens van de uniforme verdeling van lostijden
pkss	primaire kade (voor kleine schepen kade 1, voor grote schepen kade 2).
plaats	een booleanvariabele om aan te geven of kade(ss) bezet is
qt,qn	teller/noemer von Neumann ratio
qth,qnh	hulpvariabelen voor berekening van Neuman ratio
sdlt1	hulpvariabele voor printen gemdlt1
sdlt2	hulpvariabele voor printen gemdlt2
skss	secundaire of alternatieve kade (voor kleine schepen kade 2, voor grote schepen kade 1).
skwgemdlt1	variantie gemiddelde dlt1
skwgemdlt2	variantie gemiddelde dlt2
somgemdlt1	som gemiddelde dlt1 voor alle subruns
somgemdlt2	som gemiddelde dlt2 voor alle subruns
somkwgemdlt1	som van de kwadraten van gemdlt1 voor alle subruns
somkwgemdlt2	som van de kwadraten van gemdlt2 voor alle subruns

ss	schipsoort.
ta	aankomsttijd.
tb	bedieningstijd.
TIME	systeemtijd.
u1,u2	startwaarde voor de randomgenerator.
UNIFORM	procedure die getallen genereert uit een uniforme verdeling.
wf	weegfactor voor de bedieningstijd.
wr	verwijzing vanuit schip naar krij of grij.
wrss	voorrangswachtrij voor de kade (voor kade 1 krij, voor kade 2 grij).
wrnotss	alternatieve wachtrij voor de kade (voor kade 1 grij, voor kade 2 krij).

```
BEGIN COMMENT havenprobleem oude situatie kbe;
  SIMULATION
  BEGIN PROCESS CLASS generator(ss,gemtat,og,bg,u1,u2);
        CHARACTER ss; REAL gemtat,og,bg; INTEGER u1,u2;
        BEGIN WHILE TRUE DO
              BEGIN COMMENT *** wacht gedurende tussenaankomsttijd
                                volgende schip ***;
                    HOLD(NEGEXP(1/gemtat,u1));
                    COMMENT *** uitvoeren activiteit 1 ***;
                    ACTIVATE NEW schip(ss,TIME,UNIFORM(og,bg,u2));
              END;
        END***generator***;

        PROCESS CLASS schip(ss,ta,tb); CHARACTER ss; REAL ta,tb;
        BEGIN REF(kade) pkss,skss; REF(HEAD) wr;
              COMMENT *** uitvoeren activiteit 2: Bepaal van een
                          schip zijn wachtrij, zijn primaire kade
                          en zijn secundaire kade. Zet het schip in
                          zijn wachtrij en aktiveer, indien daar
                          plaats is, zijn voorrangskade of anders,
                          indien mogelijk, zijn alternatieve
                          kade ***;
              IF ss='k' THEN
                 BEGIN wr:-krij; pkss:-kade1; skss:-kade2 END
              ELSE
                 BEGIN wr:-grij; pkss:-kade2; skss:-kade1 END;
              INTO(wr);
              IF pkss.plaats THEN ACTIVATE pkss
              ELSE IF skss.plaats THEN ACTIVATE skss;
              COMMENT *** wacht op activering vanuit activiteit 5
                          ***;
              PASSIVATE;
              COMMENT *** rapporteer het vertrek van een schip ***;
```

```
      rap.vertrek(ss,TIME-ta);
      IF (rap.nk+rap.ng)>=nstop THEN ACTIVATE MAIN;
END***schip***;

PROCESS CLASS kade(ss,wrss,wrnotss,wf);
CHARACTER ss; REF(HEAD) wrss,wrnotss; REAL wf;
BEGIN COMMENT *** de procedure kortste bepaalt in een
                  wachtrij het schip met de kortste
                  bedieningstijd ***;
      REF(schip) PROCEDURE kortste(rij); REF(HEAD) rij;
      BEGIN REF(schip) boot,boot1;
            IF rij.EMPTY THEN kortste:-NONE
            ELSE
                BEGIN boot:-rij.FIRST;
                      boot1:-boot;
                      WHILE boot=/=rij.LAST DO
                      BEGIN boot:-boot.SUC;
                            IF boot.tb<boot1.tb THEN
                               boot1:-boot;
                      END;
                END;
             kortste:-boot1;
      END;
      REF(schip) schuit; BOOLEAN plaats; REAL lostijd;
      plaats:=TRUE;
      WHILE TRUE DO
      BEGIN COMMENT *** uitvoeren activiteit 3: De kade
                        probeert eerst een schip uit zijn
                        primaire wachtrij te halen en te
                        plaatsen. Als zijn primaire
                        wachtrij leeg is probeert de
                        kade een schip uit zijn secundaire
                        wachtrij te halen en te
                        plaatsen ***;
            WHILE wrss.CARDINAL+wrnotss.CARDINAL>=1 DO
            BEGIN IF NOT wrss.EMPTY THEN
                     BEGIN schuit:-kortste(wrss);
                           lostijd:=schuit.tb
                     END
                  ELSE
                     BEGIN schuit:-kortste(wrnotss);
                           lostijd:=schuit.tb*wf
                     END;
                  schuit.OUT;plaats:=FALSE;
                  COMMENT *** uitvoeren activiteit 4: Wacht
                              op einde bediening, d.w.z.
                              totdat het schip gelost is ***;
                  HOLD(lostijd);
                  COMMENT *** uitvoeren aktiviteit 5:
```

```
                                    Maak de kade vrij en
                                    verwijder het schip uit de
                                    haven ***;
                          plaats:=TRUE; ACTIVATE schuit;
                     END;
                     PASSIVATE;
                END;
           END***Kade***;

           CLASS waarnemer;
           COMMENT *** onderstaande variabelen hebben betrekking op
                     subrunresultaten ***;
           BEGIN INTEGER nk,ng;
             REAL somgemdlt1, somgemdlt2, qt, qn, qth, qnh,
                  sdlt1, sdlt2,
                  somkwgemdlt1, somkwgemdlt2, skwgemdlt1,
                  skwgemdlt2, gdk, gdg;
             REAL ARRAY gemdlt1, gemdlt2 (1:100);

             PROCEDURE vertrek(ss,dlt); CHARACTER ss; REAL dlt;
             BEGIN IF ss='k' THEN
                     BEGIN nk:=nk+1;
                          sdlt1:=sdlt1+dlt;
                     END;
                   IF ss='g' THEN
                     BEGIN ng:=ng+1;
                          sdlt2:=sdlt2+dlt;
                     END;
             END;

             PROCEDURE reportsubrun;
             BEGIN gdk:=sdlt1/nk;
                   gdg:=sdlt2/ng;
                   OUTIMAGE;
                   OUTTEXT("subrun ");OUTINT(i,2);
                   OUTIMAGE;
                   OUTTEXT("aantal kleine schepen =");
                   OUTINT(nk,4);
                   OUTIMAGE;
                   OUTTEXT("aantal grote schepen =");OUTINT(ng,4);
                   OUTIMAGE;
                   OUTTEXT("gemiddelde doorlooptijd
                          kleine schepen =");
                   OUTFIX(gdk,3,9);
                   OUTIMAGE;
                   OUTTEXT("gemiddelde doorlooptijd grote
                          schepen =");
                   OUTFIX(gdg,3,10);
```

```
        IF i>0 THEN
        BEGIN somgemdlt1:=somgemdlt1+gdk;
              somkwgemdlt1:=somkwgemdlt1+gdk*gdk;
              somgemdlt2:=somgemdlt2+gdg;
              somkwgemdlt2:=somkwgemdlt2+gdg*gdg;
        END;
  END;

  PROCEDURE initreportsubrun;
  BEGIN nk:=ng:=0;
        sdlt1:=sdlt2:=0.0;
  END;

  PROCEDURE neumannopslag;
  COMMENT *** opslaan som der doorlooptijden per
              ss per subrun na aanlooprun voor
              berekening von neumann ratio's ***;
  BEGIN gemdlt1(i):=sdlt1/nk;
        gemdlt2(i):=sdlt2/ng;
  END;

  PROCEDURE neumannberek(ss); CHARACTER ss;
  BEGIN INTEGER j,k;
        qt:=qn:=0.0;
        FOR j:=1 STEP 1 UNTIL (nsubr-1) DO
        BEGIN IF ss='k'
                 THEN qth:=gemdlt1(j)- gemdlt1(j+1)
                 ELSE qth:=gemdlt2(j)- gemdlt2(j+1);
              qt:=qt+qth*qth;
        END;
        FOR k:=1 STEP 1 UNTIL nsubr DO
        BEGIN IF ss='k'
                 THEN qnh:=gemdlt1(k)-somgemdlt1/nsubr
                 ELSE qnh:=gemdlt2(k)-somgemdlt2/nsubr;
              qn:=qn+qnh*qnh;
        END;
        OUTIMAGE;
        OUTTEXT("waarde von neumann ratio voor ");
        IF ss='k'
           THEN OUTTEXT ("kleine schepen =")
           ELSE OUTTEXT ("grote schepen =");
        OUTFIX(qt/qn, 3, 6);
  END;
  PROCEDURE eindreport;
  BEGIN OUTIMAGE;
    skwgemdlt1:=(somkwgemdlt1-
    somgemdlt1*somgemdlt1/nsubr)/(nsubr-1);
    OUTIMAGE;
    OUTTEXT("het gemiddelde van de gem. dlt kleine schepen =");
```

```
      OUTFIX(somgemdlt1/nsubr,3,8);
      OUTIMAGE;
      OUTTEXT("standaardafwijking gem dlt kleine schepen =");
      OUTFIX(SQRT(skwgemdlt1/nsubr),3,8);
      skwgemdlt2:=(somkwgemdlt2-
      somgemdlt2*somgemdlt2/nsubr)/nsubr-1);
      OUTIMAGE;
      OUTTEXT("het gemiddelde van de gem. dlt grote schepen =");
      OUTFIX (somgemdlt2/nsubr,3,8);
      OUTIMAGE;
      OUTTEXT("standaardafwijking gem dlt grote schepen =");
      OUTFIX(SQRT(skwgemdlt2/nsubr),3,9);
    END;

    somgemdlt1:=somgemdlt2:=0.0;
    somkwgemdlt1:=somkwgemdlt2:=skwgemdlt1:=skwgemdlt2:
                                                  =0.0;
  END *** waarnemer ***;

  COMMENT *** hoofdprogramma main initialisatie
              gedeelte ***;
  INTEGER i,nsa, nsubr, nsps, nstop;
  REF(generator)gen1,gen2;
  REF(HEAD)krij,grij,dummyrij;
  REF(kade)kade1,kade2;
  REF(waarnemer)rap;
  krij:-NEW HEAD;
  grij:-NEW HEAD;
  kade1:-NEW kade('k',krij,grij,1.5);
  kade2:-NEW kade('g',grij,krij,2.0);
  gen1:-NEW generator('k',5.5,3,7,876543,987654);
  gen2:-NEW generator('g',6.7,2,8,765432,654321);
  COMMENT *** lezen en opslaan van de lengte van
              de aanlooprun nsa,
              de lengte van een subrun nsps,
              het aantal subruns nsubr ***;
  OUTIMAGE;
  OUTTEXT("haven oude situatie kbe");
  OUTIMAGE;
  OUTTEXT(" ");
  OUTIMAGE;
  OUTTEXT("geef lengte aanlooprun");
  OUTIMAGE;
  nsa:=ININT;
  OUTTEXT("lengte aanlooprun=");
  OUTINT(nsa,5);
  OUTIMAGE;
  OUTTEXT("geef lengte van een subrun");
  OUTIMAGE;
```

```
        nsps:=ININT;
        OUTTEXT("lengte van een subrun =");
        OUTINT(nsps,5);
        OUTIMAGE;
        OUTTEXT("geef aantal subruns");
        OUTIMAGE;
        nsubr:=ININT;
        OUTTEXT("aantal subruns =");
        OUTINT(nsubr,3);
        OUTIMAGE;
        COMMENT *** start aanlooprun ***;
        i:=0;
        nstop:=nsa;
        rap:-NEW waarnemer;
        rap.initreportsubrun;
        ACTIVATE kade1;
        ACTIVATE kade2;
        ACTIVATE gen1;
        ACTIVATE gen2;
        COMMENT *** wacht op activering vanuit een schip
                    (einde aanlooprun) ***;
        PASSIVATE;
        rap.reportsubrun;
        COMMENT *** uitvoeren van de subruns ***;
        nstop:=nsps;
        FOR i=1 STEP 1 UNTIL nsubr DO
        BEGIN rap.initreportsubrun;
              PASSIVATE;
              rap.neumannopslag;
              rap.reportsubrun;
        END;
        rap.eindreport;
        OUTIMAGE;
        rap.neumannberek('k');
        rap.neumannberek('g');
  END;
END
```

Opmerkingen naar aanleiding van SIMULA programma:

- Het programma is zodanig opgezet dat gemakkelijk zowel de oude als de nieuwe situatie binnen de haven gesimuleerd kan worden. Door de uitvoerige parametrisatie van de gedefinieerde processen is het hoofdprogramma MAIN erg kort en flexibel. Om de nieuwe situatie te kunnen simuleren moeten de volgende wijzigingen in het programma worden aangebracht.

Met betrekking tot PROCESS CLASS schip:
oude situatie:

```
IF ss='k' THEN
        BEGIN wr:-krij; pkss:-kade1; skss:-kade2 END
ELSE
        BEGIN wr:-grij; pkss:-kade2; skss:-kade1 END;
...
...
IF pkss.plaats THEN ACTIVATE pkss
ELSE IF skss.plaats THEN ACTIVATE skss;
```

nieuwe situatie:

```
IF ss='k' THEN
        BEGIN wr:-krij; pkss:-kade1 END
ELSE
        BEGIN wr:-grij; pkss:-kade2 END;
...
...
IF pkss.plaats THEN ACTIVATE pkss;
```

Met betrekking tot het hoofdprogramma:
oude situatie:

```
kade1:-NEW kade('k', krij, grij, 1.5);
kade2:-NEW kade('g', grij, krij, 2.0);
```

nieuwe situatie:

```
dummyrij:-NEW HEAD;
kade1:-NEW KADE('k', krij, dummyrij, 1.5);
kade2:-NEW KADE('g', grij, dummyrij, 2.0);
```

De verschillen tussen de programma's voor FIFO en KBE hebben betrekking op PROCESS CLASS schip:
FIFO:

```
schuit:-wrss.FIRST;
schuit:-wrnotss.FIRST;
```

KBE:

```
schuit:-kortste(wrss);
schuit:-kortste(wrnotss)
```

Men had ook met meerdere componentklassen kunnen werken (bijv. kade1, kade2, schipg, schipk, generatork, generatorg) waardoor de procesbeschrijving per klasse eenvoudiger wordt maar uiteindelijk het gehele programma wat minder flexibel.

- Het gedrag van de stochastische variabelen kan in beide situaties identiek gemaakt worden door gebruik te maken van dezelfde random reeksen (startwaarden) van de randomgeneratoren. De waarden van de parameters U1 en U2, die gebruikt worden als startwaarden voor de randomge-

neratoren, zijn willekeurig gekozen.

- De eindrapportage wordt handmatig gedaan aangezien de t-verdeling niet standaard in SIMULA beschikbaar is.

Stap 11: *Proefruns*

Deze stap dient om globaal inzicht in het gedrag van het simulatiemodel te verkrijgen om op grond daarvan een 'definitieve' serie experimenten te plannen.

Deze stap wordt opgesplits in vier delen: de opzet van een proefrun, de resultaten, de validatie en de evaluatie.

Opzet

Omdat we geïnteresseerd zijn in de stationaire toestand kiezen we voor een lange run bestaande uit een aanlooprun gevolgd door een aantal subruns van gelijke lengte.

In eerste instantie kiezen we de lengte van de aanlooprun gelijk aan de nog te bepalen subrunlengte (zie ook figuur 4.2F en 4.3D). Op grond van de berekende bezettingsgraden (voor kade 1 gelijk aan 0.9!) en figuur 4.3D kiezen we om te beginnen een subrunlengte van 2000 componenten. (Merk op dat dit aantal overeenkomt met een aantal schepen dat in ongeveer een jaar de haven aandoet!). Verder wordt gestart met 20 subruns.

Resultaten

De resultaten van de hiervoor gespecificeerde proefrun (voor de oude situatie KBE) zijn hieronder weergegeven.

(Alleen de resultaten voor de aanlooprun en de daarop volgende 10 subruns zijn afgedrukt).

```
haven oude situatie kbe

geef lengte aanlooprun
lengte aanlooprun = 2000
geef lengte van een subrun
lengte van een subrun = 2000
geef aantal subruns
aantal subruns = 20

subrun 0
aantal kleine schepen =1096
aantal grote schepen = 904
gemiddelde doorlooptijd kleine schepen =        15.251
gemiddelde doorlooptijd grote schepen =         12.336
```

```
subrun 1
aantal kleine schepen =1094
aantal grote schepen = 906
gemiddelde doorlooptijd kleine schepen =      15.479
gemiddelde doorlooptijd grote schepen =       13.500
subrun 2
aantal kleine schepen =1109
aantal grote schepen = 891
gemiddelde doorlooptijd kleine schepen =      15.941
gemiddelde doorlooptijd grote schepen =       11.820
subrun 3
aantal kleine schepen =1099
aantal grote schepen = 901
gemiddelde doorlooptijd kleine schepen =      16.240
gemiddelde doorlooptijd grote schepen =       13.780
subrun 4
aantal kleine schepen =1129
aantal grote schepen = 871
gemiddelde doorlooptijd kleine schepen =      20.869
gemiddelde doorlooptijd grote schepen =       14.125
subrun 5
aantal kleine schepen =1142
aantal grote schepen = 858
gemiddelde doorlooptijd kleine schepen =      15.210
gemiddelde doorlooptijd grote schepen =       12.565
subrun 6
aantal kleine schepen =1108
aantal grote schepen = 892
gemiddelde doorlooptijd kleine schepen =      14.769
gemiddelde doorlooptijd grote schepen =       12.465
subrun 7
aantal kleine schepen =1061
aantal grote schepen = 939
gemiddelde doorlooptijd kleine schepen =      16.419
gemiddelde doorlooptijd grote schepen =       12.660
subrun 8
aantal kleine schepen =1122
aantal grote schepen = 878
gemiddelde doorlooptijd kleine schepen =      13.967
gemiddelde doorlooptijd grote schepen =       11.436
subrun 9
aantal kleine schepen =1089
aantal grote schepen = 911
gemiddelde doorlooptijd kleine schepen =      14.977
gemiddelde doorlooptijd grote schepen =       12.627
subrun 10
aantal kleine schepen =1160
aantal grote schepen = 640
gemiddelde doorlooptijd kleine schepen =      20.578
gemiddelde doorlooptijd grote schepen =       14.876
```

```
...
...
...
...
...
het gemiddelde van de gem. dlt kleine schepen =    16.168
standaardafwijking gem. dlt kleine schepen =       0.749
het gemiddelde van de gem. dlt grote schepen =     12.672
standaardafwijking gem. dlt grote schepen =        0.213

waarde von neumann ratio voor kleine schepen =     1.811
waarde von neumann ratio voor grote schepen =      2.070
```

Vóór het beantwoorden van de vraag van de havendirectie is het gewenst dat we in de te vergelijken situaties zoveel mogelijk gebruik maken van gemeenschappelijke randomgetallen. Daarom starten de te vergelijken situaties met dezelfde startwaarden voor de randomgenerator.

Verschillen tussen de te vergelijken situaties worden nu alleen veroorzaakt door bewust gekozen waarden van de stuurvariabelen.

Een overzicht van de resultaten voor de vier te vergelijken situaties zijn weergegeven in tabel 7.1 en 7.2.

subrun	FIFO			KBE		
nr.	oude situatie	nieuwe situatie	verschil oud-nieuw	oude situatie	nieuwe situatie	verschil oud-nieuw
0	13,4	11,9		12,3	10,5	
1	15,1	13,2	+1,9	13,5	11,4	+2,1
2	12,9	11,7	+1,2	11,8	10,5	+1,3
3	15,5	14,2	+1,3	13,7	12,1	+1,6
4	15,2	13,4	+1,8	14,1	11,8	+2,3
5	14,1	12,4	+1,7	12,5	10,9	+1,6
6	14,0	12,1	+1,9	12,4	10,7	+1,7
7	14,4	16,2	−1,8	12,6	13,6	−1,0
8	12,5	10,9	+1,6	11,4	9,8	+1,6
9	14,3	13,1	+1,2	12,6	11,2	+1,4
10	16,8	14,6	+2,2	14,8	12,9	+1,9
11	14,2	13,7	+0,5	12,7	11,8	+0,9
12	13,2	11,1	+2,1	11,9	9,9	+2,0
13	15,4	13,7	+1,7	13,4	11,8	+1,6
14	13,0	10,2	+2,8	11,9	9,5	+2,4
15	15,4	14,1	+1,3	13,0	12,1	+0,9
16	13,2	12,4	+0,8	11,9	11,0	+0,9
17	14,6	14,4	+0,2	12,8	12,2	+0,6
18	12,4	10,3	+2,1	11,2	9,5	+1,7
19	12,4	12,4	+0,0	11,4	10,9	+0,5
20	14,6	12,1	+2,5	13,0	10,7	+2,3
gem.(1-20)	14,1	12,8	+1,3	12,6	11,2	+1,4
q-waarde	2,37	2,91		2,07	2,81	

Tabel 7.1: Gemiddelde doorlooptijd grote schepen. (Per subrun in totaal (groot+klein) 2000 schepen.)

subrun nr.	FIFO oude situatie	FIFO nieuwe situatie	FIFO verschil oud-nieuw	KBE oude situatie	KBE nieuwe situatie	KBE verschil oud-nieuw
0	16,6	26,6		15,2	22,2	
1	17,1	24,0	− 6,9	15,4	20,5	− 5,1
2	17,8	32,5	−14,7	15,9	25,6	− 9,7
3	18,4	35,3	−16,9	16,2	29,6	−13,4
4	24,5	57,7	−33,2	20,8	39,5	−18,7
5	16,9	33,8	−16,9	15,2	32,8	−17,6
6	16,4	28,3	−11,9	14,7	23,5	− 8,8
7	18,9	36,7	−17,8	16,4	29,6	−13,2
8	15,4	27,1	−11,7	13,9	23,0	− 9,1
9	16,8	22,2	−5,4	14,9	18,7	−13,8
10	24,6	123,4	−98,8	20,5	72,4	−51,9
11	16,9	46,5	−29,6	15,0	57,7	−42,7
12	17,8	34,2	−16,4	15,5	27,9	−12,4
13	16,2	20,8	−4,6	15,0	17,8	− 2,8
14	21,3	67,5	−6,2	19,1	51,4	− 2,3
15	18,5	30,5	−12,0	16,7	26,1	− 9,4
16	12,6	15,2	−2,6	11,9	13,6	− 1,7
17	15,1	27,2	−12,1	13,8	22,3	− 8,5
18	14,5	23,1	−8,6	13,4	19,7	− 6,3
19	12,8	14,1	−1,3	11,7	12,7	− 1,0
20	30,9	59,9	−29,0	26,1	44,9	−18,8
gem.(1-20)	18,2	38,0	−19,8	16,1	30,5	−14,4
q-waarde	1,80	2,06		1,81	1,65	

Tabel 7.2: Gemiddelde doorlooptijd kleine schepen. (Per subrun in totaal (groot+klein) 2000 schepen.)

Evaluatie en validatie

Merk allereerst op dat de waarden van de gemiddelde doorlooptijden van de kleine schepen met name in de nieuwe situatie per subrun onderling nogal verschillen. Dit hangt samen met de zeer hoge bezettingsgraad van kade 1.

Verder blijkt uit de gepresenteerde resultaten dat de gemiddelde doorlooptijd van de schepen tijdens de aanlooprun (= subrun 0) niet duidelijk afwijkt van de overige subrunresultaten. We handhaven dus de gekozen lengte van de aanlooprun.

Binnen de grenzen van het beperkte aantal subruns blijkt dat op grond van de gevonden q-waarden (> 1,58) de subruns onderling onafhankelijk verondersteld mogen worden. Daarom zullen we de subrunlengte van 2000 handhaven. Door de formules uit hoofdstuk 4 toe te passen vinden we de resultaten uit tabel 7.3 (zie volgende pagina).

Vergelijking van de resultaten uit tabel 7.3 met de analytische resultaten uit stap 4 levert de volgende conclusies:

- Het via simulatie verkregen betrouwbaarheidsinterval voor de doorlooptijd van kleine schepen in de nieuwe situatie met FIFO rijdiscipline omvat de analytische berekende waarde (31,3).

			95%-betrouwbaarheidsinterval		
oude situatie:	kleine schepen:	FIFO	16,2 <	E(<u>dlt</u>)	< 20,2
		KBE	14,6 <		< 17,6
	grote schepen:	FIFO	13,6 <		< 14,6
		KBE	12,2 <		< 13,0
nieuwe situatie:	kleine schepen:	FIFO	26,5 <		< 49,5
		KBE	23,3 <		< 37,7
	grote schepen:	FIFO	12,1 <		< 13,5
		KBE	10,7 <		< 11,7

Tabel 7.3: 95%-betrouwbaarheidsinterval voor de doorlooptijden in de verschillende situaties.

- Het via simulatie verkregen betrouwbaarheidsinterval voor de doorlooptijd van kleine schepen in de nieuwe situatie met KBE rijdiscipline overlapt gedeeltelijk het analytisch geschatte interval.
- Beide voorgaande conclusies gelden ook met betrekking tot de grote schepen.
- De gevonden trends in de doorlooptijden van de kleine/grote schepen bij overgang van de nieuwe naar de oude situatie stemmen overeen met de kwalitatieve voorspelde trends.

De validiteit van het model voor de oude situatie wordt ondersteund door bovengenoemde conclusies (duidend op de validiteit van het model in de eenvoudigere nieuwe situatie) en het feit dat er een gering en inzichtelijk verschil bestaat in het model (programma) voor de twee situaties.

Met betrekking tot de door de havendirectie gewenste output (dit is het verschil tussen oude en nieuwe situatie) vinden we met behulp van de tabellen 7.1 en 7.2. en de formules uit hoofdstuk 4:

		95%-betr. interval voor verschil in doorlooptijd		
kleine schepen:	FIFO:	−29,9 <	E(<u>dlt</u>oud − <u>dlt</u>nieuw)	< −9,7
	KBE :	−20,6 <		< −8,2
grote schepen:	FIFO:	0,9 <		< 1,7
	KBE :	1,1 <		< 1,7

Tabel 7.4: 95%-betrouwbaarheidsinterval voor het verschil in doorlooptijd tussen de oude en nieuwe situatie.

Interessant is nu om de resultaten van tabel 7.4, welke gebaseerd zijn op 'gepaarde subruns', te vergelijken met het verschil van de subrungemiddelden voor elk van de afzonderlijke situaties, af te leiden uit tabel 7.3.

Met betrekking tot bijvoorbeeld kleine schepen, KBE geldt ruwweg:
Verschil in doorlooptijd volgens 'gepaarde' berekening: 14 ± 6
Verschil in doorlooptijd volgens 'niet-gepaarde' berekening:

$$(16 \pm 2) - (30 \pm 7) = 14 \pm 9$$

We zien dus dat door het 'paren' de nauwkeurigheid van het verschil in doorlooptijden wat toeneemt.

Stap 12: *Definitieve experimenten*

De doelstelling van het onderzoek vermeldt een onnauwkeurigheid van 10%.

De onnauwkeurigheid van de proefrunresultaten leiden we af uit tabel 7.4. Bijvoorbeeld voor kleine schepen, KBE geldt een 95% betrouwbaarheidsinterval van 12,4. Stel dat de gemiddelde waarde −14,4 gelijk is aan de verwachtingswaarde van het verschil in doorloopltijd. Een onnauwkeurigheid van 10% houdt dan in 1,44. Om de betrouwbaarheidsinterval van 12,4 terug te brengen tot 2,88 moet het aantal subruns bij benadering gelijk zijn aan $\left(\frac{12,4}{2,88}\right)^2 . 20 = 366$. Op overeenkomstige wijze kan ook voor de drie overige gevallen van tabel 7.4 het benodigde aantal subruns voor de definitieve experimenten worden geschat, waarna de definitieve experimenten kunnen worden uitgevoerd.

Stap 13: *Conclusies*

We beperken ons hier tot voorlopige conclusies op basis van de proefrunresultaten.

Uit tabel 7.3 volgt dat, zoals verwacht, KBE in alle gevallen tot een kortere doorlooptijd leidt dan FIFO.

Uit tabel 7.4. volgt dat de oude situatie voor de kleine schepen qua doorlooptijd aanzienlijk gunstiger is dan de nieuwe situatie terwijl voor de grote schepen in veel mindere mate het omgekeerde geldt. Om op grond hiervan tot een verantwoorde beslissing te komen zal een nadere kosten/baten analyse noodzakelijk zijn.

7.4 Opgaven

1 In een wasserij bevindt zich een wasmachine. Indien de wasmachine klaar is met zijn wasprogramma, wordt deze weer gevuld volgens de queuediscipline FIFO. Dit vullen gaat door totdat er geen wasgoed meer ligt, of totdat de maximum capaciteit van de machine (100 kg) overschreden wordt als de volgende partij wasgoed toegevoegd zou worden. Blijkt dit laatste het geval, dan wordt van het resterende wasgoed onderzocht of er nog een of meer partijen mee kunnen. Hierna wordt het wasprogramma gestart. Ligt er geen wasgoed meer terwijl de machine nog niet tot 60 kg. gevuld is, dan wordt gewacht tot er zoveel wasgoed is aangekomen dat deze grens wel overschreden wordt. De (uniforme) gewichtsverdeling van de pakketten wasgoed luidt als volgt:

30%	van de te wassen stukken heeft een gewicht tussen	3 – 5 kg.
50%	.. ..	5 – 10 kg.
15%	.. ..	10 – 20 kg.
5%	.. ..	20 – 30 kg.

De tijden tussen de aankomsten van partijen zijn negatief exponentieel verdeeld met een gemiddelde van 12 minuten. De wastijd (inclusief vultijd) bedraagt 1 uur.
De wasbaas zou graag met 95% betrouwbaarheid de gemiddelde doorlooptijd van het wasgoed en de fractie van de tijd dat de wasmachine stilstaat kennen voor een werkdag, waarin de wasmachine 10 keer het programma doorloopt. □

2 In een buurtwinkel komen zaterdags klanten binnen met negatief exponentieel verdeelde tussenaankomsttijden met een gemiddelde van 3 minuten. De tijd waarin ze hun boodschappen uitzoeken is uniform verdeeld tussen 1 en 6 minuten. Daarna koopt 30% van de klanten vleeswaren aan een toonbank. De helptijd hier is uniform verdeeld tussen 1 en 4 minuten. Alle klanten moeten daarna langs een kassa, waar de helptijd negatief exponentieel verdeeld is met een gemiddelde van één minuut.
De dochter des huizes moet zowel afrekenen bij de kassa als de vleeswaren snijden bij de toonbank. Zij verandert van plaats als ze bij de plaats waar ze helpt klaar is en bij de andere plaats minstens een persoon wacht. De tijd nodig voor het verplaatsen bedraagt een halve minuut.
Zaterdag is de winkel 9 uur open.
De eigenaresse van de buurtwinkel vraagt zich af of haar zoontje ook mee moet helpen.
Doorslaggevend voor haar beslissing zijn:

a de gemiddelde tijd die een klant die geen vleeswaren koopt moet wachten,

b idem voor iemand die wel vleeswaren koopt. □

3 In een kapperszaak zijn twee kappers werkzaam, elk met een eigen werkplek en wachthoek waar de nodige lektuur ligt uitgestald. Klanten, die eenmaal voor een kapper gekozen hebben, nemen in de desbetreffende wachthoek plaats.
De twee kappers hebben zich 'gespecialiseerd'.
Kapper A knipt alleen volgens een ouderwets kapsel.
Kapper B kan naast het ouderwetse kapsel ook een modieus 'snel' kapsel knippen. Keuze van kapsel is natuurlijk aan de klant.
De klanten komen de zaak binnen volgens een negatief exponentiële verdeling. De gemiddelde tussenaankomst tijd bedraagt 14 minuten.
Van de klanten is 75 procent conservatief en kiest voor het ouderwetse kapsel.
Het knippen van het ouderwetse kapsel kost 4 – 20 minuten (uniform verdeeld). Het knippen van het 'snelle' kapsel kost 8 – 12 minuten (uniform

verdeeld). Klanten, die langer dan 30 minuten moeten wachten, gaan weg. Wanneer beide wachtrijen even lang zijn of beide kappers niemand onder de kam hebben, kiest een binnenkomende klant (indien deze van een ouderwets kapsel voorzien wil worden) voor kapper A.
Het betreft hier een ideale kapperszaak, welke dag en nacht geopend is.

Gevraagd: 95% betrouwbaarheidsintervallen voor de gemiddelde wachttijd van klanten, voor het ouderwetse en het 'snelle' kapsel en het aantal weglopers voor beide kapselsoorten. □

4 In een supermarkt zijn 2 kassa's; de interaankomsttijden van de klanten bij de kassa's zijn negatief-exponentieel verdeeld met gemiddelde 1 minuut. Een klant gaat altijd in de kortste wachtrij staan. Als de wachtrijen even lang zijn, of als beide kassa's zonder klanten zijn, kiest hij willekeurig een kassa, met gelijke kansen. De bedieningstijd is uniform verdeeld tussen 1 minuut en 2.4 minuten. Wanneer op een gegeven moment een rij 2 plaatsen korter wordt dan de andere rij 'switcht' de laatste klant van de langste rij. (De kassieres hebben gemerkt dat veel klanten klagen over het lang moeten wachten). De cheffin wil nu graag weten hoeveel procent van de klanten langer dan P minuten moet wachten. □

5 In een dokterswachtkamer komen patiënten binnen volgens een negatief exponentiële verdeling met een gemiddelde tussenaankomsttijd van 5 minuten. Dertig procent van de patiënten wachten niet langer dan 10 minuten. Tijdens behandeling lopen patiënten niet weg. Wat de tijdsduren van de consulten betreft geldt het volgende:

10%	van de patiënten	2 tot 3 min.	(uniform)
30%	.. .. ..	3 tot 4 min.	..
30%	.. .. ..	4 tot 5 min.	..
20%	.. .. ..	5 tot 6 min.	..
5%	.. .. ..	6 tot 7 min.	..
2%	.. .. ..	7 tot 8 min.	..
2%	.. .. ..	8 tot 9 min.	..
1%	.. .. ..	9 tot 10min.	..

Hierbij zijn niet begrepen de tijdsduren dat de dokter gestoord wordt door zijn telefoon. Telefoongesprekken komen gemiddeld om de 6 minuten binnen met negatief exponentieel verdeelde tussenaankomsttijden. De duur van een telefoongesprek is uniform verdeeld tussen 1 en 3 minuten. Aanvragen voor telefoongesprekken die vallen binnen een bestaand telefoongesprek, vervallen. De dokter overweegt de aanschaf van een bandopname-installatie voor de telefoongesprekken, daar hij hiermee het aantal patiënten dat wegloopt zonder consult hoopt te verminderen.

Gevraagd: Een 95% betrouwbaarheidsinterval voor het verschil tussen de gemiddelde fractie patiënten die wegloopt in de oorspronkelijke situatie en die fractie in de situatie met de bandopname-installatie. □

6 Orders komen aan volgens een negatief exponentiële verdeling met gemiddelde tussenaankomsttijd van 20 minuten en moeten achtereenvolgens op machine 1 en machine 2 verwerkt worden.
De helptijd bij machine 1 is uniform verdeeld tussen 2 en 6 minuten, de helptijd bij machine 2 is uniform verdeeld tussen 4 en 6 minuten. Na verwerking op machine 1 is er een kans $p_{11} = 0,58$ dat de bewerking herhaald moet worden. De order moet dan terug naar wachtrij 1 om opnieuw bewerkt te worden waarvoor dezelfde helptijd weer nodig is. In het andere geval ($p = 1 - p_{11}$) wordt de order in wachtrij 2 geplaatst. Na verwerking op machine 2 is er een kans $p_{21} = 0,33$ dat de order terug moet naar wachtrij 1. Ook is er een kans $p_{22} = 0,25$ dat de order opnieuw door machine 2 bewerkt moet worden. De bewerkingstijden blijven hetzelfde. Met een kans $p = 1 - p_{21} - p_{22}$ is een order na bewerking op machine 2 klaar.
De queuediscipline bij de 2 machines is FIFO.
De bedrijfsleiding zou graag het 95% betrouwbaarheidsinterval voor de gemiddelde doorlooptijd van de orders kennen. □

Hoofdstuk 8

Uitwerking opgaven

Opgave 1.3.1

Wel simulatie: Bijvoorbeeld bij een machine wil men bepalen welke van de onderstaande vuistregels gehanteerd moet worden voor het afgeven van een levertijd bij orderacceptatie waarbij de verwachte levertijdoverschrijding minimaal is.

a. levertijd is constant
b. levertijd is bewerkingstijd + constante
c. levertijd is bewerkingstijd × constante

Een verdere uitbreiding van dit probleem, waarbij eveneens simulatie gewenst is, is het beoordelen van de levertijdafgifteregels in combinatie met de volgende twee prioriteitsregels voor het in bewerking nemen van de orders: kortste-bewerkingstijd-eerst en vroegste-leverdatum-eerst.

Geen simulatie:

- Als de verwachte kosten de verwachte baten ver overschrijden, bijvoorbeeld het bepalen van de grootte van de bestelserie van goedkope onderdelen met behulp van de Regel van Camp in plaats van met simulatie.
- Als er onvoldoende gekwantificeerde gegevens beschikbaar zijn, bijvoorbeeld bij een supermarkt. Hierin is het 'erg druk' bij de kassa's en men overweegt daarom een extra kassa te plaatsen. Indien men niet beschikt over geschikte kwantitatieve gegevens met betrekking tot aankomstpatronen, bedieningstijden etc. en men ook niet bereid is deze te meten is simulatie niet zinvol.
- Als het elementaire situaties betreft waarvoor analytische oplossingen bekend zijn. Bijvoorbeeld een loket dat zonder storingen functioneert met een Poisson aankomstproces van 'klanten' die worden bediend in volgorde van binnenkomst met een negatief exponentieel verdeelde bedieningstijd.

Opgave 2.9.2

a. variabelen:
Uitgangsvariabelen:

- na = aantal aankomsten bij tankstation
- nd = aantal doorrijders
- nb = aantal bediende auto's
- somwt = de som van alle wachttijden bij het tankstation van de bediende auto's

Omgevingsvariabelen:

- tussenaankomsttijden auto's bij het systeem
- bedieningstijden bij de pomp
- tijdstippen opengaan spoorbomen
- tijdstippen dichtgaan spoorbomen

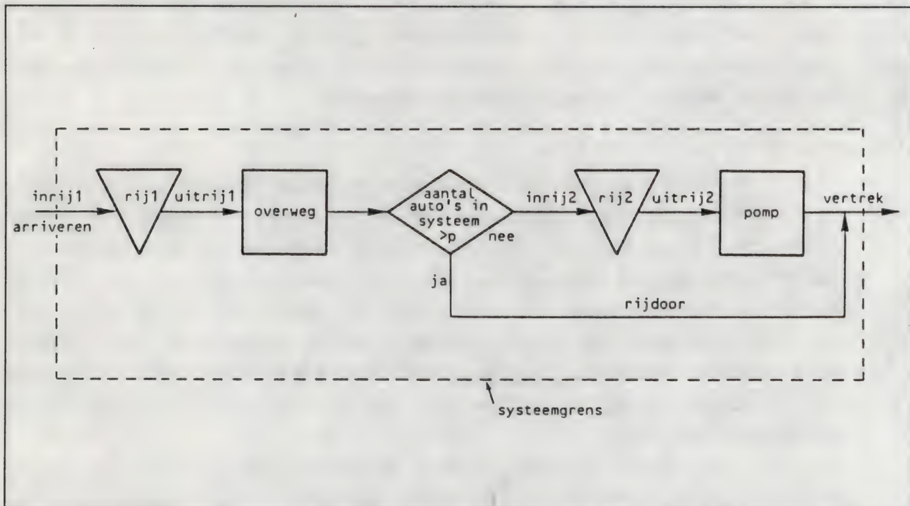
Stuurvariabelen:

- maximale lengte van de wachtrij bij de tankstation

Toestandsvariabelen:

- bo = spoorwegbomen zijn open
- pv = pomp is vrij
- l1 = lengte wachtrij voor spoorwegovergang
- l2 = lengte wachtrij bij tankstation

b. conceptueel model:



Vaste componentklassen: wachtrij, overweg, pomp
Tijdelijke componentklasse: auto

c.

activiteit	componentklasse			
	wachtrij	overweg	pomp	auto
aankomst bij overweg				x
in wachtrij voor overweg	x			x
schuif door in wachtrij	x			x
uit wachtrij voor overweg	x	x		x
passeren overweg		x		x
doorrijden bij tankstation				x
in wachtrij bij tankstation	x			x
schuif door in wachtrij	x			x
uit wachtrij bij tankstation	x		x	x
start tanken			x	x
tanken klaar			x	x
vertrek			x	x
bomen dicht		x		
bomen open		x		

activiteitsklasse	veranderingen in toestandsvariabelen
aankomst auto bij overweg	–
auto in wachtrij voor overweg	l1:=l1 + 1
auto schuift door in wachtrij	–
auto uit wachtrij voor overweg + passeert overweg	l1:=l1 – 1
auto rijdt door bij tankstati	–
auto in wachtrij bij tankstation	l2:=l2 + 1
auto schuift door in wachtrij tankstation	–
auto uit wachtrij bij tankstation + start tanken	l2:=l2 – 1, pv:=FALSE
tanken klaar + vertrek	pv:=TRUE
spoorbomen gaan dicht	bo:=FALSE
spoorbomen gaan open	bo:=TRUE

d.

Wat zijn de eventklassen in ons geval? M.a.w.: door welke gebeurtenissen verandert de toestand van ons systeem? (m.a.w.: waardoor veranderen de waarden van de toestandsvariabelen bo, pv, l1 en l2 van ons systeem?)

Antwoord: bij
- het opengaan der bomen (bo, l1, l2 en/of pv) o
- een vertrek (pv en/of l2) v
- een aankomst (l1 of l2 of pv) a
- het sluiten der bomen (bo) s

Verbale eventbeschrijvingen (*essentieel!*).

Aankomst : Indien spoorweg gesloten gaat auto in wachtrij1 staan, anders passeert auto overweg. Indien de pomp vrij is wordt deze bezet, anders wordt wachtrij2 één langer, tenzij wachtrij2 daardoor langer zou worden dan p auto's, dan zal de auto doorrijden.

Vertrek : De pomp komt vrij. Indien er nog auto's in wachtrij2 staan, dan gaat de voorste auto uit deze wachtrij naar de pomp. De eventuele aanwezige overige auto's schuiven één plaats op in wachtrij2. Indien er geen auto's in wachtrij2 staan, gebeurt er niets.

Bomen sluiten : Bomen gaan dicht.

Bomen openen : Bomen gaan open. Indien er auto's in wachtrij1 staan, dan gaan de auto's uit wachtrij1 achtereenvolgens naar wachtrij2, zolang lengte wachtrij2 $< p$, zodra wachtrij2 $\geq p$ rijden ze door. Indien de pomp vrij is en de *eerste* auto uit wachtrij1 bij wachtrij2 aankomt, dan wordt de pomp door deze auto bezet, anders wordt wachtrij2 één langer tenzij lengte wachtrij2 $> p$: dan rijdt de eerste auto (en alle overige auto's uit wachtrij1) door.

e.

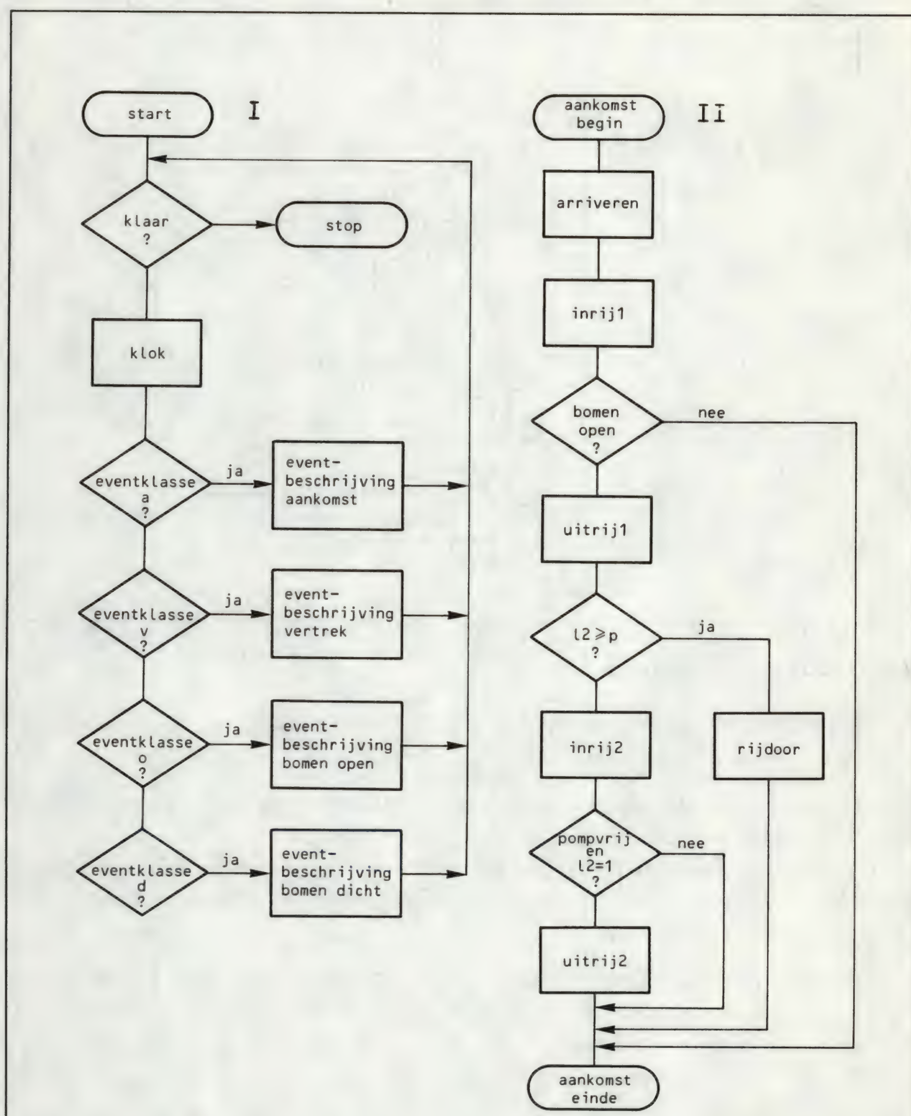
Uitgaande van de eventbeschrijvingen en de geformuleerde activiteitsklassen vinden we:

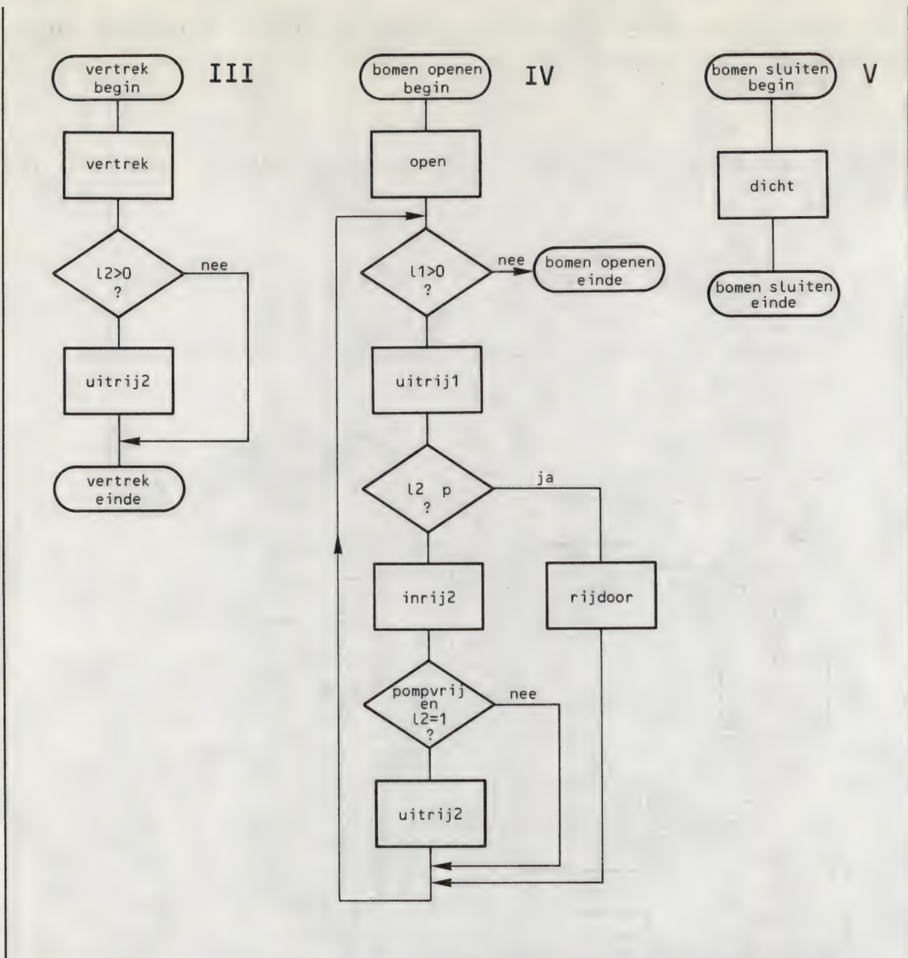
act. kl.	verandering tst. var.	voorwaarde op syst. niv	eventkl. waartoe act. kl mogelijk behoort	verandering omg. var.
arriveren		t=vat	a	vat:= t+ NEGEXP(v,u) (tijdstip volg. aank.)
inrij1	l1:=l1+1		a	
(schuifdoor)			o	
uitrij1	l1:=l1−1	bo,l1>0	a o	
rijdoor		l2>=p	a o	
inrij2	l2:=l2+1	l2<p	a o	
(schuifdoor)			v	
uitrij2	l2:=l2−1 pv:=FALSE	pv,l2>0	a o v	vvt:=t+ UNIFORM(a,b,u) (tijdstip volg. vertr.)
vertrek	pv:=TRUE	t=vvt	v	
open	bo:=TRUE	t=vot	o	vst:=t+25 (volg.tijdst. bom. sluit.)
dicht	bo:=FALSE	t=vst	s	vot:=t+5 (volg. tijdst. bom. open)

(Opmerking kolom 3 en 5 behoren tot een volgende stap die in een later hoofdstuk behandeld zal worden).

f.

De stroomschema's van de eventbeschrijvingen zijn weergegeven in de figuren I t/m V.





Toelichting op de vijf figuren:

Fig. I: Globaal stroomschema van de eventbeschrijving.

Fig. II: Bij de eventklasse 'aankomst' is slechts één component van de componentklasse auto betrokken.

Het stroomschema is verkregen door de verschillende activiteitsklassen behorend tot de evenklasse a in de volgorde van tabel e onder elkaar te zetten.

Fig. III: Bij de evenklasse 'vertrek' kunnen 1 tot maximaal p (max. lengte wachtrij2) componenten van componentklasse 'auto' betrokken zijn. Eén component is altijd de vertrekkende auto. Indien $l2 \neq 0$ dan gaat de eerste auto uit wachtrij2 naar de pomp (begin bediening) en schuiven de overige auto's in wachtrij2 één plaats op, anders is het event 'vertrek' al meteen afgelopen.

Fig. IV: Bij de eventklasse 'bomen openen' is één component van de componentklasse 'overweg' betrokken; 0, 1, ... componenten van de componentklasse 'auto' (leden van wachtrij1); 1 of 2 componenten 'wachtrij' (wachtrij1 en/of wachtrij2) en één component 'pomp'.

Dit stroomdiagram is op dezelfde wijze tot stand gekomen, als figuur II. We zien dat de eventklassen 'aankomst' en 'bomen openen' een groot stuk gemeen hebben.

Fig. V: De enige activiteit die plaatsvindt bij de eventklasse 'bomen sluiten' is het dichtgaan van de bomen. Hierbij is alleen 1 component van de componentklasse 'overweg' betrokken.

Opgave 2.9.8

Het ligt voor de hand storingen (defecten) als een apart soort orders op te vatten. Het optreden van een storing komt er op neer dat de bewerkingstijd van de order die op dat moment bewerkt wordt met de duur van de storing verlengd wordt. Verder is het van belang dat tijdens de bewerking van een order er meer dan één storing kan optreden.

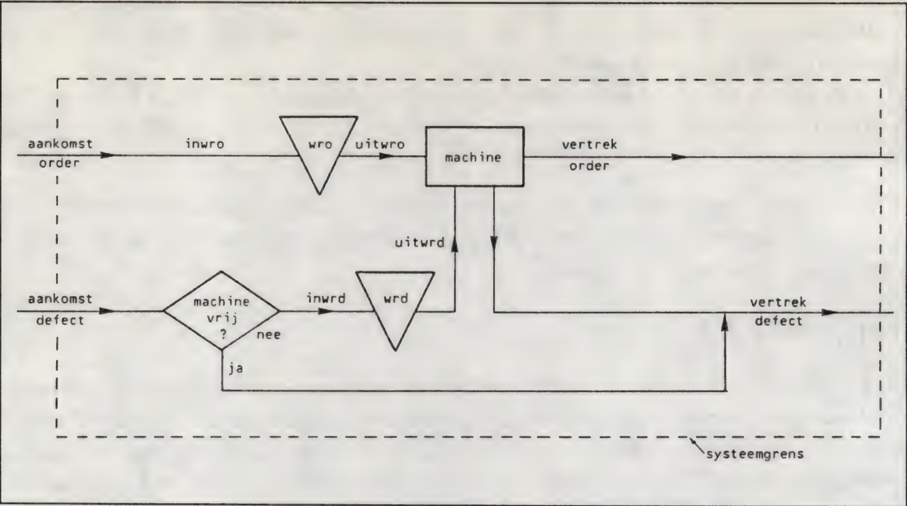
a. Uitgangsvariablelen: nob = aantal orders bewerkt tot nu toe
somdrlptd = som van de doorlooptijden van de bewerkte orders tot nu toe

Omgevingsvariabelen: vato = volgende aankomsttijd order, afhankelijk van de orderaankomstsnelheid vo,
vats = volgende aankomsttijd storing, afhankelijk van de storingsaankomstsnelheid vs

Stuurvariabelen: ogbto,bgbto = onder- resp. bovengrens bedieningstijd van order
ogbts,bgbts = onder- resp. bovengrens 'bedieningstijd' van storing

Toestandsvariabelen: lwro = lengte wachtrij orders
VRIJ = machine vrij

b. conceptueel model:



Vaste componentklassen : machine, wachtrij, waarnemer(v/m), aankomstgenerator
Tijdelijke componentklassen : order, storing

c.

activiteit	order	storing	machine	wrs	wro	ord.gen	stor.gen
A1 aankomst order						x	
A2 in wro	x				x		
(A0 schuif door)	x				x		
A3 uit wro + naar machine	x		x		x		
A4 start bewerking	x		x				
A5 einde bewerking + van machine	x		x				
A6 aankomst storing							x
A7 in wrs		x		x			
(A11 schuif door)		x		x			
A8 uit wrs + naar machine		x	x	x			
A9 start reparatie		x	x				
A10 einde reparatie + van machine		x	x				

d.

Act. klasse	verandering in toestandsvar.
A0	
A1	
A2	`lwro:=lwro+1`
A3	`lwro:=lwro-1`
A4	`VRIJ:=FALSE`
A5	`VRIJ:=TRUE`
A6	
A7	`IF VRIJ=FALSE THEN lwrs:=lwrs+1`
A8	`lwrs:=lwrs-1`
A9	`mach(ine)stand:="B"`
A10	`mach(ine)stand:="V"`
A11	

e.

Act. klasse	Componentklasse					
	ogen	sgen	wro	wrs	machine	order	storing
A0			x				
A1	x						
A2						x	
A3					x		
A4					x		
A5					x		
A6		x					
A7				x			
A8					x		
A9					x		
A10					x		
A11				x			

f. Stroomdiagram

Het stroomdiagram staat op de volgende pagina.

Opgave 4.5.1

Geen duidelijke aanloopverschijnselen door lage bezettingsgraad. Daardoor wordt de overeenkomstige korte gemiddelde rijlengte $\frac{r}{2(1-r)} = 0,5$ en bijbehorende beperkte wachttijd en doorlooptijd snel bereikt ook bij het starten vanuit een leeg systeem.

BESTURING
creëer + activeer WAARNEMER1
WAARNEMER1
KLAAR
creëer + activeer WRD1
WRD1
wacht op A8
A11
creëer + activeer WRO1
WRO1
wacht op A3
A0
creëer + activeer MACHINE1
MACHINE1
lengte WRO1 >0 ?
nee
wacht op A2
ja
A3
A4
wacht tot TIME = einde actuele bewerkingstijd
lengte WRO1 >0 ?
nee
ja
A8
A9
wacht tot TIME = einde duur actuele defect
A10
A5
creëer + activeer OGEN1
OGEN1
wacht tot TIME = aankomsttijd order
A1
order
A2
KLAAR
creëer + activeer DGEN1
DGEN1
wacht tot TIME = aankomsttijd defect
A6
DEFECT
machine bezet ?
nee
ja
A7
KLAAR
genoeg subruns ?
wacht tot TIME = einde werkdag
rapportage per subrun
eind-rapportage
STOP

Opgave 4.5.2

Lage bezettingsgraad: Systeem is dan regelmatig leeg. Hierdoor wordt de toestand van het systeem op willekeurig moment niet sterk beïnvloed door de toestand die een langere tijdsduur hieraan vooraf ging. Aangezien een leeg systeem als 'ontkoppelpunt' kan worden opgevat resulteert dit in een korte subrunlengte.

Hoge bezettingsgraad: Systeem zelden leeg. Gemiddelde rijlengte groot. Er treden weinig 'ontkoppelpunten' op. Toestand van het syteem sterk beïnvloed door voorafgaande toestanden gedurende een relatief lange periode. Dus hoge subrunlengte om 'onderlinge onafhankelijkheid' te bereiken.

Opgave 4.5.3

Het herhalen (met andere randomgetallen) van runs die steeds starten vanuit een leeg systeem waarbij het 'aanloopdeel' (waarvan de lengte nog nader bepaald moet worden) niet in de meting wordt meegenomen.

Opgave 5.5.2

Maximaal toegestane waarde van p is gelijk aan de arraygrootte waarin de aankomstmomenten worden opgeslagen, in dit geval 100.

Opgave 6.8.2

```
BEGIN
  COMMENT stuurvariabelen;
  INTEGER p; COMMENT inlezen maximale aantal wachtpl. op het
                                                emplacement;
  OUTIMAGE; p:=ININT; COMMENT inlezen waarde p;
  BEGIN
  COMMENT hulpvariabelen;
    BOOLEAN stopprogram;
    REAL inf; COMMENT oneindig;
    INTEGER u; COMMENT startwaarde randomgenerator;
    INTEGER maxauto; COMMENT aantal auto's per subrun;
    REAL ARRAY rij2(1:p);
    COMMENT omgevingsvariabelen;
    REAL v, a, b, vat, vvt, vst, vot;
    CHARACTER ek; COMMENT eventklasse;
    COMMENT toestandsvariabelen;
    INTEGER l1, l2;
    BOOLEAN bo, pv;
    COMMENT uitgangsvariabelen;
    REAL somwt;
    INTEGER na, nd, nbediend;
```

```
COMMENT proceduredeclaraties;
PROCEDURE klok; COMMENT bepaling tijdstip en type van
                        eerstvolgende event;
BEGIN
    COMMENT t=min(vat, vvt, vot, vst);
    IF vat<t THEN BEGIN ek:='a'; t:=vat; vat:=inf; END;
    IF vvt<t THEN BEGIN ek:='v'; t:=vvt; vvt:=inf; END;
    IF vot<t THEN BEGIN ek:='o'; t:=vot; vot:=inf; END;
    IF vst<t THEN BEGIN ek:='s'; t:=vst; vst:=inf; END;
END;

PROCEDURE naarp;
BEGIN INTEGER i;
    somwt:=somwt+t-rij2(1);
    l2:=l2-1; COMMENT opschuiven resterende auto's in
                                                wachtrij2;
    IF l2>0 THEN  FOR i:=1 UNTIL l2 DO rij2(i):=rij2(i+1);
    pv:=FALSE;
    vvt:=t+UNIFORM(a,b,u); COMMENT tijdstip eerst volgende
                                                   vertrek;
END;

PROCEDURE gatanken;
BEGIN
    l1:=l1-1;
    IF l2>=p THEN nd:=nd+1
         ELSE BEGIN
              l2:=l2+1; rij2(l2):=t;
              IF l2=1 AND pv THEN naarp
              END
END;

COMMENT initialisatie;
v:=.25; a:=2.0; b:=5.0;
l1:=0; l2:=0;
bo:=pv:=TRUE; stopprogram:=FALSE;
somwt:=0.0; na:=nd:=nbediend:=0;
vat:=NEGEXP(v,u);
vst:=25; COMMENT bij begin simulatie zijn bomen juist
                                          open gegaan;
inf:=10**99; u:=918273; maxauto:=500;
vvt:=vot:=inf;
t:=inf;

COMMENT simulatie;
WHILE NOT stopprogram DO
BEGIN klok;
    IF ek='a' THEN
```

```
        BEGIN
            na:=na+1;
            vat:=t+NEGEXP(v,u);
            l1:=l1+1;
            IF bo THEN gatanken;
        END;
        IF ek='v' THEN
        BEGIN
            pv:=TRUE;
            nbediend:=nbediend+1;
            IF l2>0 THEN naarp;
        END;
        IF ek='o' THEN
        BEGIN
            bo:=TRUE;
            vst:=t+25;
            WHILE l1>0 DO
            BEGIN
                gatanken
            END
        END;
        IF ek='s' THEN
        BEGIN
            bo:=FALSE;
            vot:=t+5;
        END;
        IF (nbediend + nd) = maxauto THEN stopprogram:=TRUE;
    END;
    COMMENT druk resultaten af;
  END;
END
```

Opgave 6.8.8

Componentklasse	Classtype	Reden
waarnemer	CLASS	geen 'driehoek' of 'rondje' in procesbeschr. componentklasse, geen wachtrij, geen wachtrijelement
wachtrij	HEAD	wachtrij
machine	PROCESS CLASS	'driehoek' en 'rondje' in procesbeschr. componentklasse
aankomstgenerator	PROCESS CLASS	'rondje' in procesbeschr. componentklasse
order	LINK CLASS	geen 'driehoek' of 'rondje' in procesbeschr. componentklasse, wel wachtrijelement
storing	LINK CLASS	idem

Het SIMULA programma:

```
BEGIN SIMULATION BEGIN
CLASS waarnemer; BEGIN .........END;
PROCESS CLASS m; BEGIN ..........END;
PROCESS CLASS aankgen(type,iai,ogbt,bgbt,u);
    CHARACTER type; REAL iai, ogbt, bgbt; INTEGER u; NAME u;
    BEGIN .............END;
LINK CLASS order(bt); REAL bt; BEGIN ..........END;
LINK CLASS storing(duur); REAL duur; BEGIN ........END;

REF(waarnemer) marlies; REF(m) mach;
REF(HEAD) wro, wrs;
COMMENT activeren vaste componenten;
marlies:-NEW waarnemer; wro:-NEW HEAD; wrs:-NEW HEAD;
mach:-NEW m; ACTIVATE mach;
ACTIVATE NEW aankgen('o',6.5,5,10,123456);
ACTIVATE NEW aankgen('s',48,10,20,789012);
HOLD (480); COMMENT simulatie betreft 1 werkdag van 8 uur;
marlies.rapport;
END
```

Uitwerking van de diverse componentklassen:

```
LINK CLASS order(bt); REAL bt; COMMENT bt=bewerkingstijd;
BEGIN
    REAL at; COMMENT aankomsttijd at is nodig voor het
                     bepalen van doorlooptijd dlt;
    at:=TIME; INTO (wro);
    IF wro.CARDINAL = 1 AND mach.vrij THEN ACTIVATE mach;
END;

LINK CLASS storing(duur); REAL duur;
BEGIN
    IF mach.vrij = FALSE THEN INTO(wrs);
END;

PROCESS CLASS m; COMMENT machine;
BEGIN BOOLEAN vrij; REF(order) job; REF(storing) stor; vrij:=TRUE;
WHILE TRUE DO
BEGIN
    IF wro.CARDINAL = 0 THEN PASSIVATE;
    vrij:=FALSE;
    WHILE wro.CARDINAL > 0 DO
    BEGIN
        job:-wro.FIRST; job.OUT;
        HOLD(job.bt);
        WHILE wrs.CARDINAL > 0 DO
        BEGIN
            stor:-wrs.FIRST; stor.OUT;
```

```
            HOLD(stor.duur);
        END;
        marlies.drlptd(TIME-job.at);
    END;
    vrij:=TRUE;
    END;
END;

PROCESS CLASS aankgen(type,iai,ogbt,bgbt,u);
CHARACTER type; REAL iai, ogbt, bgbt; INTEGER u; NAME u;
BEGIN
WHILE TRUE DO
    BEGIN
        HOLD(NEGEXP(iai,u));
        IF type = 'o' THEN NEW order(UNIFORM(ogbt,bgbt,u))
                      ELSE NEW storing(UNIFORM(ogbt,bgbt,u));
    END;
END;

CLASS waarnemer;
BEGIN   REAL somdrlptd; INTEGER nob;
PROCEDURE drlptd(dlt); REAL dlt;
    BEGIN
        nob:=nob+1; somdrlptd:=somdrlptd+dlt;
    END;
PROCEDURE rapport;
    BEGIN
        OUTIMAGE; OUTTEXT ("Gemiddelde doorlooptijd orders: ");
        IF nob > 0 THEN OUTFIX (somdrlptd/nob,2,10);
                   ELSE OUTTEXT ("geen verwerkte orders");
    END;
nob:=0; somdrlptd:=0.0;
END;
```

Hoofdstuk 9

Literatuur

Baker, K.R., Bertrand, J.W.M., 'An investigation of due-date assignment rules with constrained tightness', *Journal of Operations Management*, Vol. **1**, no. 3 (1981)

Birtwistle, G.M., Dahl, O.-J., Nijgaard, K., *SIMULA Begin*, Studentenlitteratur, Lund, 1984.

Cameron, 'An overview of JSD', *IEEE Transact. on softw. engng.*, Vol. **SE-12**, no. 2, pp. 224-240, 1986.

Harmon, P., King, D., *Expert systems*, John Wiley & Sons, New York, 1985.

Hillier, F.S., Yu, O.S., *Queueing tables and graphs*, North Holland, New York, 1981.

Jackson, M., *System Development (JSD)*, Prentice-Hall, Englewood Cliffs, N.J., 1983.

Jackson, M., *Systeemontwikkeling volgens JSD*, (Nederlandse vertaling van Jackson, 1983), Academic Service, Den Haag, 1987.

Kerbosch, J.A.G.M., Sierenberg, R.W., *Discrete simulatie met behulp van Algol, Fortran, PL/1*, Samson, Alphen a/d Rijn-Brussel, 1973.

Kleijnen, J.P.C., *Statistical tools for simulation practitioners*, Marcel Dekker, New York 1986/1987.

Law, A.M., Kelton, W.D., *Simulation Modelling and Analysis*, McGraw-Hill, New York, 1982.

Neelamkavil, F., *Computer Simulation and Modelling*, John Wiley & Sons, New York, 1987.

Papazoglou, M.P., Georgiadis, P.I., Maritsas, D.G., 'An outline of the programming language SIMULA', *Comp. Lang.* **9**, 2, pp. 107-131, 1984.

Pidd, M., *Computer Simulation in Management Science*, John Wiley & Sons, New York, 1984.

Shannon, H.G., *Systems Simulation: the art and science*, New York, 1975

Spriet, J.A., Vansteenkoste, G.C., *Computer-aided Modelling and Simulation* Academic Press, London, 1982.

Swartz, R., *Modelling and Simulation on microcomputers*, The Soc. for computer simulation, La Jolla, California, USA, 1984.

Veld, in 't Veld, J., *Analyse van organisatieproblemen*, Elsevier, Amsterdam/Brussel, 1984.

Wagner, H.M., *Principles of operations research*, Prentice-Hall, Englewood Cliffs, 1975.

Problemen oplossen met de computer - R.G. Dromey
Inleiding tot het programmeren, deel 1 en 2 - J.J. van Amstel e.a.
Het Groot Pascal Spreukenboek - Henry F. Ledgard e.a.
Software engineering, het bouwen van grote programma's - I. Sommerville
JSP Jackson struktureel programmeren - Henk Jansen
JSP Uitwerkingenboek, JSP Procedureboek - Henk Jansen

PROGRAMMEERTALEN

Aspecten van programmeertalen - J.J. van Amstel & J.A.A.M. Poirters
Programmeertalen, een inleiding - J.J. van Amstel e.a.
Colloquium programmeertalen - red. J.A.A.M. Poirters & G.J. Schoenmaker
BASIC - EIT-serie, deel 3
Cursus BASIC, een practicum handleiding voor BASIC op de PRIME - R. Bloothoofd e.a.
Een programmeercursus in BASIC - Nok van Veen & René Wissing
Cursus Pascal - A. van der Sluis & C.A.C. Görts
Cursus eenvoudig Pascal - A. van der Sluis & C.A.C. Görts
Inleiding programmeren in Pascal - C. van de Wijgaart
Modula-2 - E. Verhulst
Systeemontwikkeling met Ada - Grady Booch
Cursus COBOL - A. Parkin
Cursus FORTRAN 77 - J.N.P. Hume & R.C. Holt
De programmeertaal C - L. Ammeraal
Flitsend Forth - Alan Winfield
Logisch LOGO - Auke Sikma
Programmeren in LISP - L.L. Steels
Micro-PROLOG, programmeren in logica - K.L. Clarke & F.G.McGabe
Inleiding PROLOG - W. Burnham & A. Hall

BESTANDSORGANISATIE, DATABASE EN GEGEVENSANALYSE

Bestandsorganisatie - R.J. Lunbeck & F. Remmen
Database, een inleiding - C.J. Date
Databases, grondslagen voor de logische structuur - F. Remmen
SQL in de praktijk - H.B. Eilers e.a.
Het SQL Leerboek - Rick F. van der Lans
Gegevensanalyse - R.P. Langerhorst

INFORMATIE-ANALYSE EN SYSTEEMONTWERP

Inleiding systeemanalyse en systeemontwerp - W.S. Davis
Systeemontwikkeling zonder zorgen - Paul T. Ward
Systeemontwikkeling volgens SDM - H.B. Eilers
Samenvatting SDM - Pandata
Systeemontwikkeling volgens JSD - Michael Jackson
Informatie-analyse volgens NIAM - J.J.V.R. Wintraecken
Information engineering - J. Blank
Het ontwerpen van interactieve toepassingen en computernetwerken - J.A. Scheltens
EDP Audit - C. de Backer
Management informatiesystemen - G.B. Davis & M.H. Olson
Prototyping, een instrument voor systeemontwerpers - red. T. Hoenderkamp & H.G. Sol
Het ontwikkelen van informatiesystemen met prototyping - R. Vonk
Simulatie, een moderne methode van onderzoek - S.K.T. Boersma & T. Hoenderkamp

EXPERTSYSTEMEN EN KUNSTMATIGE INTELLIGENTIE

Kunstmatige intelligentie - Patrick H. Winston
Expertsystemen - Henk de Swaan Arons & Peter van Lith
Ontwikkelingen in expertsystemen - red. A. Nijholt & L.L. Steels

THEORETISCHE INFORMATICA EN SYSTEEMPROGRAMMATUUR

Systeemprogrammatuur - H. Alblas
Vertalerbouw - H. Alblas e.a.

OPERATIONELE RESEARCH

Operationele research - Y.M.I. Dirickx e.a.
Lineaire programmering als hulpmiddel bij de besluitvorming - S.W. Douma

[illegible] opgelost met de gegevens [illegible]
[illegible] voor [illegible] van Amstel
[illegible]
[illegible] Baker
[illegible]

PROGRAMMEERTALEN

[illegible] Amstel [illegible] A. M. [illegible]
[illegible] Tuyl
[illegible] Amstel [illegible] en A. J. [illegible]
[illegible]
[illegible] BASIC [illegible]
[illegible] BASIC [illegible] Kloek Wissels
[illegible]
[illegible]
[illegible] Klijnsma
[illegible]
[illegible] Groot [illegible]
[illegible]
[illegible] Wolff
[illegible] Anderson
[illegible]
[illegible]
[illegible]
[illegible] Geert
[illegible]

GEGEVENSBANKEN, DATABASE EN GEGEVENSANALYSE

[illegible]
[illegible]
[illegible] Renout
[illegible]
[illegible]
[illegible]

[illegible] EN SYSTEEMONTWERP

[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]
[illegible]

EXPERTSYSTEMEN [illegible]

[illegible]
[illegible]
[illegible]

THEORETISCHE INFORMATICA EN SYSTEEMPROGRAMMATUUR

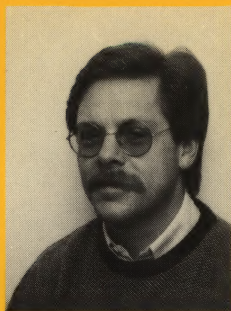
[illegible] R. Adams
[illegible]

OPERATIONS RESEARCH

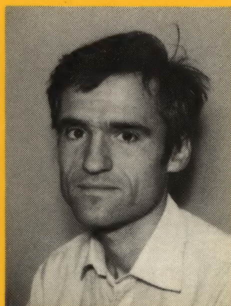
Operations research [illegible]
[illegible]

INFORMATIE [illegible]

Academic Service, Postbus 81, 2870 AB Schoonhoven [illegible]



Paul Griep studeerde elektrotechniek aan de H.T.S. en aan de Technische Universiteit te Enschede met als specialisatie bio-informatica. Daar verrichtte hij ook zijn promotieonderzoek, waarbij het ontwikkelen van een simulatiemodel voor het verklaren van het elektrofysiologisch gedrag van spieren centraal stond. Daarna was hij vier jaar werkzaam op het gebied van de ziekenhuisautomatisering te Tilburg, ondermeer als projectleider bij de ontwikkeling van een geautomatiseerd afsprakensysteem. Sedert 1984 is hij werkzaam bij de vakgroep 'Bestuurlijke Informatie Systemen en Automatisering' van de faculteit Bedrijfskunde van de T.U. Eindhoven en werkt mee aan het onderwijs onder andere op het gebied van simulatie en het onderzoek op het gebied van methoden en hulpmiddelen voor het onwikkelen van informatiesystemen.



Simme Douwe Flapper studeerde natuurkunde en biologie aan de Rijksuniversiteit te Utrecht. Na zijn promotie op het gebied van 'liquid crystals' aan de Katholieke universiteit te Nijmegen werkte hij drie jaar als senior systeemontwerper/database administrator voor het directoraat-generaal voor de Milieuhygiëne. Vanaf 1985 is hij werkzaam bij de faculteit Bedrijfskunde van de Technische Universiteit Eindhoven, waar hij naast het verzorgen van onderwijs o.a. op het gebied van simulatie, meewerkt aan het ontwikkelen van informatiesystemen ten behoeve van goederenstroom- en afdelingsbeheersing.

Met behulp van discrete simulatie kan men inzicht verkrijgen in het gedrag van (reële) systemen waarvan de toestand slechts op discrete tijdstippen verandert, bijvoorbeeld ter ondersteuning van (bestuurlijke) beslissingen of als hulpmiddel bij het ontwerpen van nieuwe systemen.

Het eerste 'algemene' deel van dit boek behandelt hoe men uitgaande van een probleemdefinitie komt tot een logische specificatie van een simulatiemodel. Het dynamisch gedrag hiervan kan hierbij op drie wijzen worden vastgelegd: met behulp van de event-, de activiteiten- of de procesbeschrijving. De nadruk wordt gelegd op de procesbeschrijving (object-georiënteerde aanpak).

In het tweede deel van dit boek wordt aangegeven hoe de logische specificatie geïmplementeerd kan worden in een taal als SIMULA. Hiertoe wordt eerst een niet-technische inleiding in deze inmiddels 'klassieke' taal gegeven. Centraal hierbij staat het zogenoemde class-concept dat kenmerkend is voor object-georiënteerde talen (SIMULA, SMALLTALK, S84). Hierna worden een tweetal voorbeelden tot in detail uitgewerkt. De hier gehanteerde wijze van modelspecificeren en implementeren kan tevens als basis dienen voor het gebruik van methoden als 'Jackson Structured Development' ten behoeve van het ontwikkelen van 'gewone' informatiesystemen.

Dit boek komt voort uit collegedictaten van de Technische Universiteit Eindhoven waarbij getracht is het geschikt te maken voor het HBO-onderwijs en voor zelfstudie.

ISBN 90 6233 272 2
NUGI 852